About This Book

This book is a technical reference for the programming tools in the OS/2 Toolkit. It is intended for use by application programmers creating programs using OS/2 system functions.

Who Should Read This Book

The OS/2 Tools Reference is intended for programmers knowledgeable in at least one programming language in which OS/2 programs can be written.

How This Book is Organized

- Introduction

  This chapter contains information about all chapters in the book and the function of the tools.

- PACK/UNPACK and PACK2/UNPACK2

  This chapter describes the pack, unpack, pack2, and unpack2 tool, which reduces and expands the size of files.

- Dialog Editor

  This chapter describes the Dialog Editor and how it is used to create and modify dialog boxes.

- Executable File Header Utility (EXEHDR)

  This chapter describes the Executable File Header Utility (EXEHDR) that displays and modifies the contents of an executable-file header.

- Font Editor

  This chapter describes the Font Editor and how it is used to define fonts and edit character width.

- Forwarded Entry Point (FWDSTAMP)

  This chapter describes the FWDSTAMP tool, which adds entry points, called *forwarders* to a dynamic link library file.

- Icon Editor

  This chapter describes the Icon Editor, which lets you create your own icons, pointers, and bit maps and save them for use by applications.

- Managing Import Libraries (IMPLIB)

  This chapter describes the IMBPLIB tool, which creates import libraries used to link dynamic-link libraries with applications.

- Link for Object and Library Files (LINK386)

  LINK386 is a tool used to combine object files and standard library files into a single file: an executable file, a dynamic-link library, or a device driver.

- Make Message File (MKMSGF)

  This chapter describes the MKMSGF tool, which reads the input message file and creates an output message file.

- Make Template File (MKTMPF)

  This chapter describes the MKTMPF tool, which creates template repository files from text input files.

- MAP File to SYM File Creator (MAPSYM)

  This chapter describes the MAPSYM tool, which creates .SYM files from .MAP files.

- View and Set Program Type For Executable File (MARKEXE)

  This chapter describes the MARKEXE tool, which enables you to view and set the program type for an executable file.

- Message Segment Binder (MSGBIND)

  This chapter describes the MSGBIND tool, which binds a message segment to an executable program.

- Object Utility/2 Description

  This chapter describes how Object Utility/2 provides a facility for registering Workplace Shell classes, creating instances of Workplace Shell classes, and modifying instances of Workplace Shell classes.

- Program Maintenance Utility Program (NMAKE)

  This chapter describes the NMAKE tool, which automates the process of building an application from project files.

- Quick Information (KwikINF)

  This chapter describes the KwikINF tool, which provides a quick and convenient method of accessing information in online documents from anywhere on the desktop.

- Resource Compiler

  This chapter describes the OS/2 Resource Compiler an application-development tool that lets you add application resources , such as message strings, pointers, menus, and dialog boxes, to the executable file of your application.

- Workplace Class List

  This chapter discusses the Workplace Class List tool, which creates Workplace Shell object classes.

Double-Byte Character Set (DBCS)

Throughout this publication, there are references to specific values for character strings. These values are for the Single-Byte Character Set (SBCS). When using the Double-Byte Character Set, notice that one DBCS character equals two SBCS characters.

Documentation Conventions

Throughout this library of documents, the following conventions distinguish the different elements of text:

plain text	Function names, structure names, data types names, message names, enumerated types, and constant names.
Initial capitalization	Key names, push buttons, checkboxes, radio buttons, group-box controls, drop-down list box, dialog windows, spin buttons, combo-boxes, SLE and MLE fields.
CAPITALS	File names and error codes.
`monospace`	Programming examples and user input at the command line prompt or into an entry field.
bold	Action bar choices and menu items.
italics	Parameters, structure fields, titles of documents, and first occurrences of words with special meaning.

Introduction

The *Tools Reference* is a technical reference for the tools in the OS/2 Toolkit. The following is a brief description of each chapter in this book and the function of each tool.

Dialog Editor

Purpose

The Dialog Editor draws dialog boxes and controls on the screen so you can see what they look like when used by your application.

Description

You can use the Dialog Editor to create and modify dialog box and modify the controls and text within dialog boxes. As you create the dialog box and its controls, you see them on the screen as the user sees them when using your program. You can place each dialog box and its controls where you want them on the screen. You also can test the dialog box before you incorporate it into your application. The Dialog Editor saves the dialog boxes in a ASCII-text format files that have .DLG extensions. For more information see Dialog Editor.

Executable File Header Utility (EXEHDR)

Purpose

The Executable File Header Utility (EXEHDR) displays and modifies the contents of an executable-file header.

Description

EXEHDR generates an output listing showing the contents of the file header and information about each object or segment in the file. Options are provided that let you change values in the file header. For more information see Executable File Header Utility (EXEHDR).

Font Editor

Purpose

Use the Font Editor to design and save fonts for use in your applications.

Description

The Font Editor enables you to edit an enlarged version of each character in an editing window, using the mouse to switch the enlarged representation of pels to black or white.

A font is a set of alphanumeric characters, punctuation marks, and other symbols that share a common typeface design and line weight. An application loads a font from a dynamic-link library file (DLL file). For more information see Font Editor.

Forwarded Entry Point (FWDSTAMP)

Purpose

FWDSTAMP adds entry points, called forwarders, to a dynamic link library file.

Description

Forwarders point to API functions or other exported code or data. They contain and import reference so that the final target address of the forwarded entry is contained in a different module. A forwarder might be called an imported export. For more information see Forwarded Entry Point (FWDSTAMP).

Generate Message Catalog Utility (GENCAT)

Purpose

One of the requirements for internationalization of programs is that messages be displayed in the language of the user. This requirement is satisfied by producing versions of the messages which are translated into all supported languages.

The OS2 C library messaging support is based on the messaging support described in the X/Open XPG4 specification. It consists of functions for extracting messages from message catalog files (catopen, catgets, catclose) and utilities for producing message catalog files (gencat, mkcatdef). The functions are documented in the *C Library Reference* and the utilities are documented in this book.

The GENCAT utility processes a message source file and produces a catalog file usable by the C library messaging functions.

Description

GENCAT creates the message catalog (usually *.cat) from a message source file (usually *.msg) or standard input. You can specify any number of message source files. A message source file is a text file that contains messages consisting of a message number followed by the message text. See the Preprocess Message Source File Utility (MKCATDEF) for using symbolic message identifiers in your message source file.

For more information on GENCAT, see Generate Message Catalog Utility (GENCAT).

Icon Editor

Purpose

The Icon Editor lets you create icons, pointers, and bit maps and save them for use by applications.

Description

You can customize Icons, pointers, and bit maps for your application.

An application can use an icon to represent a minimized standard window. For example, for an application that lists telephone numbers you could create a telephone icon. For more information see Icon Editor.

Managing Import Libraries (IMPLIB)

Purpose

IMPLIB creates import libraries used to link dynamic-link libraries with applications. The import library tells the application where to find the dynamic-link libraries at run time. See Managing Import Libraries (IMPLIB) for an explanation of import libraries and why they should be used with dynamic-link libraries.

Description

IMPLIB takes a module definition file (.DEF) as input. For each export definition in the .DEF file, IMPLIB generates a corresponding import definition.

The .LIB file generated by IMPLIB is used as input to LINK386, which creates an executable (.EXE) file. The .LIB file provides LINK386 with

information about imported dynamic link functions.

Quick Information (KwikINF)

Purpose

KwikINF provides you with a quick and convenient method of accessing information in online documents stored in the OS/2 BOOKSHELF from anywhere on the desktop, with the exception of DOS or WIN-OS/2* sessions.

Description

You can open a dialog with KwikINF by pressing a user-selectable hot key after starting KwikINF. The KwikINF window also allows you to initiate searches for text strings in on-line documents of your choice. For more information see Quick Information (KwikINF).

Link for Object and Library Files (Link386)

Purpose

Link386 is used to combine object files and standard library files into a single file: an executable file, a dynamic-link library, or a device driver.

Description

LINK386 can produce dynamic-link libraries (.DLL) and device Drivers (.SYS), in addition to executable files (.EXE). For additional information, see Link for Object and Library Files (LINK386).

MAP File to SYM File Creator (MAPSYM)

Purpose

MAP File to SYM File Creator (MAPSYM) is a program that creates .SYM files from .MAP files.

Description

.SYM files are used by the kernel debugger for symbolic debugging. MAPSYM must be run from the directory in which the file to be converted is located. For more information see MAP File to SYM File Creator (MAPSYM).

View and Set Program Type For Executable File (MARKEXE)

Purpose

The MARKEXE program enables you to view and set the program type for an executable file. For applications running on OS/2 for SMP Version 3, MARKEXE enables you to set the MPUNSAFE bit, which forces the application to be run in uniprocessor mode.

Description

Use MARKEXE with the OS/2 Linear Executable Linker (Link386) or the OS/2 Segmented Executable Linker (LINK) to change or set the program type of programs you have created, and to set or unset the MPUNSAFE bit. For more information see View and Set Program Type For Executable File (MARKEXE).

Preprocess Message Source File Utility (MKCATDEF)

Purpose

The Generate Message Catalog Utility (GENCAT) does not accept symbolic message identifiers; you must use the MKCATDEF utility if you want to use symbolic message identifiers in you message source file for use with GENCAT.

Description

MKCATDEF processes a message source file containing symbolic identifiers and produces the following output:

- The SYMBOLNAME.H file, containing statements that equate symbolic identifiers with the set numbers and message ID numbers assigned by MKCATDEF. You must include the SYMBOLNAME.H file in your application program to associate the symbolic names to the set and message numbers assigned by MKCATDEF.
- A new message source file containing message numbers instead of symbolic message identifiers. This output is suitable for input to GENCAT.

After running MKCATDEF, you can use symbolic names in an application to refer to messages.

For more information on MKCATDEF, see Preprocess Message Source File Utility (MKCATDEF).

Make Message File (MKMSGF)

Purpose

There are two ways that the output message file can be used:

- Selected messages can be bound to the message segment of an executable file using the MSGBIND program.
- Messages can be accessed directly from the output message file. For more information see Make Message File (MKMSGF).

Description

The Make Message File (MKMSGF) program reads the input message file specified and creates an output message file that DosGetMessages uses to display messages.

Make Template File (MKTMPF)

Purpose

MKTMPF creates template repository files from text input files.

Description

A template repository (also referred to as a *repository file* or *repository* in this document) is a binary file used by the operating system's Error Logging Facility to find error descriptions, causes, and actions in various message files.

The first step in creating a repository file is to create a text input file for MKTMPF using any text editor. MKTMPF reads and validates this file, reporting any errors or warnings, before translating it to binary data and creating a repository file. For more information see Make Template File (MKTMPF).

Message Segment Binder (MSGBIND)

Purpose

Message Segment Binder (MSGBIND) is a program that binds a message segment to an executable program.

Description

Message Segment Binder reads an input file that specifies the executable files to modify. For each executable file, MSGBIND specifies which message files to scan, and for each message file, it specifies which messages to include in the executable file. For more information see Message Segment Binder (MSGBIND).

Program Maintenance Utility (NMAKE)

Purpose

Program Maintenance Utility (NMAKE) automates the process of updating project files and can be used to make backups, configure data files, and run programs when data files are modified.

Description

Program Maintenance Utility (NMAKE) compares the modification dates for one set of files (the target files) with those of another set of files (the dependent files). If any dependent files have changed more recently than the target files, NMAKE executes a series of commands to bring the targets up-to-date. For more information see Program Maintenance Utility Program (NMAKE).

Object Utility/2

Purpose

Object Utility/2 provides a facility for registering Workplace Shell classes, creating instances of Workplace Shell classes, and modifying instances of Workplace Shell classes.

Description

The Object Utility/2 has attributes that can be set or modified. The attributes modify the behavior of the objects to enable or disable copying, deletion, and other attributes. For more information see Object Utility/2 Description

---

Data Compression (PACK/UNPACK and PACK2/UNPACK2)

Purpose

You can use PACK and UNPACK to compress and expand files. The options, parameters, and function for PACK2 are identical to PACK.

Note: PACK and PACK2 are shipped with the OS/2 Toolkit. UNPACK and UNPACK2 are shipped with OS/2.

2 **Description**

Data Compression (PACK) is a tool that reduces the size of a file by compressing its data. Decompression (UNPACK) works in reverse and allows you to expand files. You can use PACK for a single file or group of files, thereby reducing the disk space required for your OS/2 application. The only difference between PACK and PACK2 is that PACK2 has a better compression algorithm. For more information see PACK/UNPACK and PACK2/UNPACK2.

Resource Compiler

Purpose

The OS/2 Resource Compiler (RC) is an application-development tool that lets you add application resources, such as message strings, pointers, menus, and dialog boxes, to the executable file of your application.

Description

The Resource Compiler is primarily intended to prepare data for OS/2 applications that use functions such as WinLoadString, WinLoadPointer,WinLoad Menu, and WinLoadDlg. For more information see Resource Compiler.

Trace Customizer (TRCUST)

Purpose

OS/2 provides a mechanism by which developers may dynamically apply tracepoints in their module at run time. This method eliminates all overhead of tracing when tracing is disabled. It also allows the developer to add tracepoints without modifying source code. This reduces the possibility that adding a tracepoint will induce errors into one's code. OS/2 needs a binary file, for each module being dynamically traced, which defines the tracepoints for the module.

Description

The Trace Customizer (TRCUST) converts tracepoint definitions from a trace source file (TSF) into dynamic tracepoints for the trace definition file (TDF), and into formatting rules in the trace format file (TFF). For more information see Dynamic Trace Customizer (TRCUST).

Workplace Class List

Purpose

Workplace Class List creates a workplace object class and an instance of a workplace object class.

Description

Workplace objects are constructed using the SOM protocol and are called Predefined, Subclass or Replaced object classes. For a definition of Predefined, Subclass and Replaced classes, see Workplace Class List.

Dialog Editor

You use the *Dialog Editor* to create and modify dialog boxes, and to create and modify the controls and text within dialog boxes. As you create the dialog box and its controls, you see them on the screen as the user will see them when your program is run. You can place each dialog box and its controls where you want them on the screen. In addition, you can test the dialog box before you incorporate it into your application.

Each dialog box and control can have either an integer identifier or a symbolic identifier that equates to an integer identifier. You use the identifier in your application to refer to the dialog box or control. If you intend to use symbolic identifiers in your application, you must enter the symbolic and integer identifiers in an include file. If you do not use symbolic names, the Dialog Editor supplies an integer identifier for each control and for the dialog box itself. You can use the Dialog Editor to create the include file, or you can use a text editor to create the include file before using the Dialog Editor.

It is good programming practice to plan the resources that your application will use and to choose a naming and numbering convention for the symbolic or integer identifiers before you create them. Keep the include file separate from other include files used by your application. The Dialog Editor will use only #define statements from an include file. It ignores everything else it finds in the file.

Although the Dialog Editor draws dialog boxes and controls on the screen so you can see what they look like when used by your application, it does *not* save them as graphics. Instead, the Dialog Editor saves them in an ASCII-text format file that has a .DLG extension. Refer to the dialog template section of this chapter.

The Dialog Editor also creates a compiled form of the .DLG file in a resource file with a .RES extension. The .DLG and .RES files can

contain more than one dialog box. The resource file can contain other application resources, such as icons, bit maps, and string tables. It is attached to the executable (.EXE) file of the application during the compile and link process.

Designing Dialog Boxes

Dialog boxes should be designed to clearly identify the information that the user is required to complete. The following are a few Common User Access* guidelines:

- Lay out the controls in columns, starting at the upper-left corner, for left-to-right or top-to-bottom scanning.
- Vertically and horizontally align selection and entry fields so that the cursor moves in a straight line.
- Arrange the controls in the sequence in which the user would complete them.
- If there are only a few entry fields, locate them at the top of the dialog box.
- Make groups of controls obvious by use of group boxes and white space.
- Align group boxes, where possible. Group boxes can be extended to the right to line up with other group boxes.
- Use field identifiers to identify the purpose of single and multiple groups of choices.

Creating a Dialog Box

To run the Dialog Editor, select **Dialog Editor** from the **Development Tools** folder. The main window appears, displaying the menu bar choices **File**, **Edit**, **Control**, **Arrange**, **Options**, and **Help**. On line help that tells you how to use the editor is available on most Dialog Editor windows.

To create a new dialog box, start with either of the following methods:

- Select **New Dialog** from the **Edit** menu. The editor opens new files with the extensions .RES and .DLG. This also opens a new include file.
- Select **New** from the **File** menu. This opens new files with the extensions .RES and .DLG. You can open a new include file or an existing one.

When you edit a dialog box, the names of the resource and include files are shown in the title bar of the Dialog Editor. If you are editing a new file that has not yet been named or saved, **(Untitled)** appears in the title bar in place of a name. If **(Untitled)** appears in the title bar in place of a name, there are unsaved changes.

The **Dialog Box ID** field appears in the status area. A default integer number is supplied in the entry field. Type a symbolic identifier for the dialog box, such as MYDIALOG. Tab to the integer field and type the integer number. Press Enter to place them both in the include file.

The new dialog box appears in the lower-left corner of the editor screen enclosed by a border. The border contains eight small squares called drag handles, which allow you to change the width and height of the selected item. This indicates that the dialog box is selected for editing. If you are creating a new dialog box, the dialog is automatically selected; at all other times, before you edit the dialog box or a control, you must click on it to select it.

To continue creating the new dialog box, follow these steps:

1. Make the dialog box larger by clicking on one of its drag handles with the left mouse button and dragging until the box is the size you want it to be. This can be done in one operation by clicking on the upper-right corner of the border and dragging diagonally upwards and to the right.

   Information about the item you are editing is displayed in the **Selected Item Status** box in the left half of the status area. As you move the shadow box, the x-y-coordinates change. These are the coordinates of the origin of the dialog box relative to the origin of the window. The cx-cy-coordinates are the width and height of the dialog box. The symbolic identifier is also shown.

2. Select **Styles** from the **Edit** menu. The Dialog Box Styles pop-up window appears.

3. Click on the text entry field in the status area, and then type the dialog box title (for instance, `Sample dialog box`) into the field.

4. Press Enter and the title appears at the top of your dialog box.

You can reposition the entire dialog box by moving the pointer inside the top area enclosed by the border, holding the left mouse button down, and dragging the shadow box across the screen. When the shadow box is in the position where you want the dialog box to appear, release the mouse button. The dialog box appears in that position. Alternatively, you can move the dialog box using the keyboard arrow

keys. You can reposition the dialog box at any time during the edit.

Using a Grid

Before you start adding controls to the dialog box, you might want to first select the grid option to make laying out your dialog easier.

You can use a mouse to place controls in a dialog box and to move the controls in line with each other. However, you can more accurately position the controls by using the keyboard arrow keys or mouse after grid values have been set.

The **Settings-change** dialog lets you set the number of character spaces (in dialog units) by which you can move dialog boxes and controls when using the Dialog Editor.

To set the grid size, follow these steps:

1. Select **Settings** from the **Arrange** menu. The **Settings-change** dialog is displayed. The initial grid setting for both *x* and *y* is 1 unit.

2. Change the *x*-setting to 10 and the *y*-setting to 5. Click on **OK**.

The horizontal (x) and vertical (y) values are in dialog units. A horizontal dialog unit is 0.25 of the standard character size. A vertical dialog unit is 0.125 of the standard character size. For example, if you move a control to the left or the right (using the mouse or keyboard arrow keys) with **x** set at 20, it moves in steps of twenty dialog units.

When you subsequently position dialog boxes or controls, the objects move by the specified number of dialog units on an invisible grid. Large values make it easier to align controls, while small values allow you to position controls in the dialog box more precisely.

Now that the grid is in place, you are ready to start adding controls.

Adding Controls

The control menu lists, in alphabetic order, all the controls that you can put in a dialog box. To add controls, follow these steps:

1. Select a control from the **Control** menu or click on an icon on the Control Palette at the right side of the window.

   The pointer becomes a small plus sign (+) in a square. The center marks the position where the lower-left corner of the border for the control will be set.

2. Click the mouse to position the control.

3. A dialog might appear (depending on the type of control) in which you must enter data or check preferences to define the control. Complete this and close the dialog.

For an example of adding controls in a typical dialog, see Example.

You might want to test the dialog.

For detailed descriptions of individual controls and how they work, see the individual controls in the on line help (while using the Dialog Editor) by following these steps:

1. Select **Help Index** from the **Help** menu (or press F1 and select **Help Index**).

2. Select **Options** or press Alt-O.

3. Select **Contents** or press Ctrl-C.

4. Select **Control Menu** for an alphabetic list of controls, or select **Control Palette** for the icons as they appear on the Control Palette.

5. Select the control you want to read about.

Arranging Controls

The Arrange menu allows you to arrange and align controls in a logical and easy-to-understand layout.

Align	Aligns controls along an edge.
Even spacing	Evenly spaces controls
Same size	Sets controls to the same size.
Push buttons	Arranges push buttons.
Order groups	Displays the **Groups-order** dialog, so you can change the order of controls and groups.
Settings	Displays the **Settings-change** dialog, so you can change the grid and spacing constants.

Ordering Control Groups

This option allows you to gather controls into groups and to change the order in which the tab keys and arrow keys move the selection cursor around the controls.

When you use group boxes to group controls, always create the group box before the controls that are to go inside it.

It is good practice to put group markers around all separate groups of controls, including putting a marker before the first control in the list.

The list box shows the order in which the selection cursor moves between the controls when the user presses the arrow and tab keys. (The coordinate position of a control when displayed in the dialog box does not affect the order.) Initially, the controls are listed in the order in which they were created.

There are three functions involved in grouping controls:

- Setting Group Markers
- Setting Tab Markers
- Moving Control Order

Setting Group Markers

To set up groups in a dialog that has various types of controls, follow these steps:

1. Select **Order Groups** from the **Arrange** menu. The **Groups - order** dialog is displayed.
2. Click on the first radio button in the list box.
3. Click on the **Group Marker** push button. A group marker is now displayed between the **Text** control and the first radio button in the list.
4. Scroll down the list and click on the first push button in the list. Click on the **Group Marker** push button. This has organized your controls into groups of text, radio buttons, check boxes, and push buttons.

Setting Tab Markers

After setting group markers, you will want to set tab-stops. The controls marked with an asterisk already have tab-stops.

To make the tab-stop at only the first control in each group, delete the tab-stops from the second and third radio button and check box, following these steps:

1. Click on the second radio button in the list to mark it.
2. Click on the **Delete Tab** push button.
3. Repeat the above steps for the third radio button, and then perform the same operation for the second and third check box in the

list. When this is complete, press Enter.

Moving Control Order

You can move controls in the list and then see during testing how the changes affect the movement of the cursor. To change the position of a control in the list, follow these steps:

1. Click on the name of the control to select it.
2. Position the pointer in the list where you want the name to appear. The pointer changes shape to a short horizontal line when it is over a place where you can insert the name.
3. To insert the control name, click the mouse button.

After grouping controls, you might want to test or edit the dialog, or enter additional controls.

Selecting Color and Font

The Presentation Parameters dialog allows you to select the color and font for individual controls or for an entire dialog box.

You can select all of the following:

- Foreground Color
- Background Color
- Foreground Color Highlight
- Background Color Highlight
- Disabled (greyed out) Foreground Color
- Disabled (greyed out) Background Color
- Font Size
- Font Name

To set presentation parameters, follow these steps:

1. Select a control or the dialog box.
2. Select **Presentation Parameters** from the **Edit** menu.
3. Type the number, from 1 to 255 parts of each color, in the appropriate fields.
4. Type the font size and name, if you want to change the default, in the last two fields.
5. Select **OK** or press Enter to close the dialog.

You might now want to test the dialog.

Using the Options Menu

On the Options menu, a check mark next to each option shows whether it is selected (on) or not (off).

To toggle your selection of options on and off, use the following functions of the Options menu:

- Select **Test mode** to test the dialog.
- Select **Hex mode** to toggle between hexadecimal and decimal display of ID Values of symbols.
- Select **Translate mode** to toggle translate mode on and off.
- Select **Enable 2.x styles** to use controls and their styles which are specific to OS/2 2.x, but not prior releases.
- Select **Show status area** to toggle display of the status area on and off.

Example

The control menu lists, in alphabetic order, all the controls that you can put in a dialog box. The sample dialog is Sample Dialog Template File. To add controls for a sample dialog, follow these steps:

1. Select **Text** from the **Control** menu or select a control by clicking on its icon on the Control Palette at the right side of the window.

   The pointer becomes a small plus sign (+) in a square. The center marks the position where the center of the control will be.

2. Position the pointer inside the dialog box near the upper-left corner and click the mouse.

3. Type `Student Level:` in the **Text** entry field. Observe that the next sequential integer is supplied in the **Symbol** entry field. Press Enter.

4. Replace the symbol with **ID_GRAD** and press Enter.

   The Dialog editor assigns the next integer to the symbolic identifier you entered and places it in the include file. This is another technique for entering symbolic identifiers.

5. To view or change the include file at any time, select **Symbols** from the **Edit** menu. The **Symbols** dialog appears.

   The symbolic and integer identifier for the dialog box and the text control are displayed in the list box. The dialog allows you to add, delete, and change the identifiers and to view the hexadecimal equivalents of the integers.

   Select the **OK** push button to remove the dialog and register any changes. Select **Cancel** if you have not made any changes.

6. In your dialog box, the static control is not large enough for you to see all the text. To remedy this, click on the text, and a border appears around it. Drag the right-hand edge of the border to the right to enlarge the field.

   When you release the mouse button, you should be able to see all the text. When a control has a border around it, it is selected and you can use a shadow box to position it, as you did with the dialog box.

7. To add another control, select **Radio Button** from the **Control** menu and position the cursor just beneath the **Student Level** text. Press Enter.

8. Type `Elementary` in the **Button Text** entry field and press Enter. Drag the right edge of the border that surrounds the radio button until you can see all of the text.

9. Select **Radio Button** again and type `Intermediate` in the **Text** entry field. Position this radio button below the first one.

10. Select **Radio Button** again and type `Advanced` in the **Text** entry field. Position this radio button below the other two.

11. Select **Group Box** from the **Control** menu. Position the cursor to the right of the column of radio buttons and press Enter.

12. Type `Media` in the **Text** entry field and press Enter to title the group box.

13. Click on the lower-right corner of the group box border and drag it diagonally down and to the right to enlarge it. The bottom of the group box border should be lower than the last of the radio buttons, and the right-hand side of the group box should be almost at the far right of the dialog box. This is to make room for a group of check boxes that will go inside the group box.

    When you use group boxes to group controls, you always create the group box before the controls that are to go inside it.

14. Select **Check Box** from the **Control** menu. Position the cursor inside the group box in line with the first radio button in the list, and click the mouse.

15. Type `TextBooks` in the **Button Text** entry field and press Enter. Enlarge the border of the check box until all of the text is displayed.

16. Select **Check Box** again and position the cursor below the first check box. Type `Video` in the **Text** entry field and click Enter. Enlarge the check box border until all of the text is displayed.

17. Select **Check Box** again and position the cursor below the previous two check boxes. Type `Diskettes` in the **Text** entry

field.

In the left-hand side of the dialog box, you should now have a column of radio buttons with a heading of **Student Level**, and on the right a group box with a heading of **Media** that contains three check boxes.

18. Finally, add three push buttons to the dialog box. Select **Pushbutton** from the **Control** menu. Position the cursor in the lower-left side of the dialog box and click the mouse. Type `OK` in the **Text** entry field and press Enter.

19. Position another push button to the right of the first one (in the lower middle of the dialog box) and type `Cancel` in the **Text** entry field.

20. Select a third push button and position it to the right of the second. Type `Help` in the **Text** entry field.

The dialog box and its controls are now complete.

Try selecting each of the controls, and observe the information in the **Selected Item Status**. It holds information about each control that you edit.

You might now want to test the dialog box.

Changing the Dialog Box

To change the properties of a dialog box or a single control, use the following functions of the **Edit** menu:

- Select **New Dialog** to create another dialog box in the same resource file. Your existing dialog box will stay in memory.
- Select **Select Dialog** to switch to another open dialog box.
- Select **Symbols** to define symbols.

Eight of the editing functions require that you first select the control to be edited. The selected control will appear in the *drag window*, surrounded by eight dots, one in each corner and one at the midpoint of each side.

The following functions require that a control must first be selected:

- Select **Cut** to cut a control you would like to move or delete.
- Select **Copy** to copy to the clipboard a control you would like to duplicate elsewhere in the same dialog or in another dialog.
- Select **Paste** to place a control you have marked with **Cut** or **Copy**.
- Select **Clear** to erase a control.
- Select **Duplicate** to create another control in this dialog box that is identical to the selected control.
- Select **Styles** to define the style of the selected control.
- Select **Presentation parameters** to select the colors and fonts.
- Select **Size to text** to adjust the size of an entry field to the text inside.

Testing the Dialog Box

To test the dialog box, select **Test Mode** from the **Options** menu. The dialog box is displayed as it will appear to the user in a program. In test mode, you can select controls, and their appearance changes in the same way as they do in an application. To return to work mode, click on **Test Mode** again to de select it.

If you want to make changes, you can edit the dialog box.

Ending an Edit Session

To end the edit session, select **Close** from the system pull-down menu. You see prompts for the file names of the files you want to save.

If you want to edit the same file the next time you use the editor, select **Open** from the **File** menu.

Dialog Templates

The Dialog Editor creates an ASCII text file that has the file-name extension .DLG. The compiled form of this file, created using the Resource Compiler, has the file-name extension .RES.

The .DLG file contains a series of statements, collectively termed a *dialog template* , that define each dialog box and each control in each dialog box. The statement for each dialog box contains the data required to create it, namely its class, size, position, window text, and any other special information required for the window.

Normally, the template consists of a dialog box window followed by the controls contained within it, which are child windows.

The first statement in the template is the DLGINCLUDE statement, which specifies the file name of the include file.

The next statement is the DLGTEMPLATE statement, which specifies the symbolic identifier of the dialog box (MYDIALOG). The DLGTEMPLATE statement also specifies any loading and memory options. The actual dialog template is contained within the first BEGIN and last END statement. There is a CONTROL statement for each of the controls in the dialog box. The CONTROL statement is a general statement that is followed by parameters that further specify the control, such as:

- Text, where appropriate. For example, the text **OK** is defined for one of the push buttons.
- Application-defined symbolic or integer identifiers for each control. Your application uses the identifier to track the responses from controls. For example, ID_NULL is the identifier of the text control.
- The types and positions of the various controls. For example, the group box control is a control window of window class WC_STATIC. The **Cancel** and **Help** push buttons are of window class WC_BUTTON.
- The appearance and operation of the dialog box and its controls, which are specified in detail by combinations of style parameters. For example, the check boxes have a class style of BS_CHECKBOX, and radio buttons have a class style of BS_RADIOBUTTON. You can also specify appropriate WS_* styles.

If necessary, you can use a text editor to edit the .DLG file, for example, to *fine-tune* the dialog template produced by the dialog box editor. You can even use a text editor to produce your own .DLG file. The Dialog Editor uses the general CONTROL statement with window classes and control styles to define controls.

You can use the CONTROL statement in the same way to define your controls, or you can use any of several predefined control statements that give you the same result. For example, the predefined control statement PUSHBUTTON gives you a WC_BUTTON class window with default styles of BS_PUSHBUTTON and WS_TABSTOP.

The predefined controls are described in the following sections.

AUTOCHECKBOX Statement

The **AUTOCHECKBOX** statement creates an automatic-check-box control. The control is a small rectangle (check box) that contains a check when the user selects it. The specified text is displayed to the right of the check box. A check appears in the square when the user first selects the control and disappears the next time the user selects it. The **AUTOCHECKBOX** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the text, identifier, coordinates, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify the style, the default style is BS_AUTOCHECKBOX and WS_TABSTOP.

AUTORADIOBUTTON Statement

The **AUTORADIOBUTTON** statement creates an automatic-radio-button control. This control is a small circle with the given text displayed to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control also removes the selection from any other automatic-radio-button controls in the same group. When the user selects the button again, the control removes the highlight before sending a message. The **AUTORADIOBUTTON** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the text, identifier, coordinates, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_AUTORADIOBUTTON.

CHECKBOX Statement

The **CHECKBOX** statement creates a check-box control. The control is a small rectangle (check box) that has the specified text displayed to the right. The control highlights the rectangle and sends a message to its parent window when the user selects the control. The **CHECKBOX** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the text, identifier, coordinates, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_CHECKBOX and WS_TABSTOP.

COMBOBOX Statement

The **COMBOBOX** statement creates a combination-box control. This control combines a list-box control with an entry-field control. It allows you to place the selected item from a list box into an entry field. The **COMBOBOX** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the text, identifier, coordinates, dimensions, and attributes of a control window. The predefined class for this control is WC_COMBOBOX. If you do not specify a style, the default style is CBS_SIMPLE, WS_GROUP, WS_TABSTOP, and WS_VISIBLE.

CONTAINER Statement

The **CONTAINER** statement creates a container control within a dialog window. The container control is a visual component that holds objects. The **CONTAINER** statement defines the identifier, coordinates, dimensions, and attributes of a container control. The predefined class for this control is WC_CONTAINER. If you do not specify a style, the default style is WS_TABSTOP, WS_VISIBLE, and CCS_SINGLESEL.

Example

This example creates a container control at position (30,30) within the dialog window. The container has a width of 70 character units and a height of 25 character units. Its resource ID is 301. The default style CCS_SINGLESEL has been overridden by the style specification CCS_MULTIPLESEL. The default styles WS_TABSTOP and WS_GROUP are both in effect, though only the latter is specified.

```
#define IDC_CONTAINER    301
#define IDD_CONTAINERDLG 504
DIALOG "Container", IDD_CONTAINERDLG, 23, 6, 120, 280, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
     CONTAINER   IDC_CONTAINER, 30, 30, 70, 200, CCS_MULTIPLESEL |
                       WS_GROUP
  END
```

DEFPUSHBUTTON Statement

The **DEFPUSHBUTTON** statement creates a default push button control. The control is a round-cornered rectangle containing the given

text. The rectangle has a bold outline to represent that it is the default response for the user. The control sends a message to its parent window when the user chooses the control. The **DEFPUSHBUTTON** statement, which you can use only in a **DIALOG** or **WINDOW** statement, lets you define the coordinates, dimensions, and attributes of the default push button control. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_PUSHBUTTON, BS_DEFAULT, and WS_TABSTOP.

EDITTEXT or ENTRYFIELD Statement

The **EDITTEXT** or **ENTRYFIELD** statement creates an entry-field control. This control is a rectangle in which the user can type and edit text. The control displays a pointer when the user selects the control. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the Backspace and Delete keys. By using the mouse or the direction-arrow keys, the user can select the character or characters to delete or select the place to insert new characters.

The **EDITTEXT** or **ENTRYFIELD** statement defines the text, identifier, coordinates, dimensions, and attributes of a control window. The predefined class for this control is WC_ENTRYFIELD. If you do not specify a style, the default style is ES_AUTOSCROLL and WS_TABSTOP. The **EDITTEXT** control statement is identical to the **ENTRYFIELD** control statement. Use the **EDITTEXT** or **ENTRYFIELD** statement only in a **DIALOG** or **WINDOW** statement.

FRAME Statement

The **FRAME** statement defines a frame window. The statement defines the title, identifier, position, and dimensions of the frame window, as well as the window style. The **FRAME** statement is most often used in a **WINDOWTEMPLATE** statement, and typically, only one **FRAME** statement is used. The **FRAME** statement, in turn, typically contains at least one **WINDOW** statement that defines the client window belonging to the frame window.

The frame window has no default style. You must use the *framectl* field to define additional frame controls, such as a title bar and system menu, to be created when the frame window is created. If the text field is not empty, the statement automatically adds a title-bar control to the frame window, whether or not you specify the FCF_TITLEBAR style. Frame controls are given default styles and control identifiers based on their class. For example, a title-bar control receives the identifier FID_TITLEBAR.

The **FRAME** statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a **FRAME** statement contains one **WINDOW** statement.

Example

This example creates a standard frame window, with title bar, a system menu, minimize and maximize boxes, and a vertical scroll bar. The **FRAME** statement contains a **WINDOW** statement defining the client window belonging to the frame window.

```
WINDOWTEMPLATE 1
BEGIN
    FRAME "My Window", 1, 10, 10, 320, 130, 0,
            FCF_STANDARD | FCF_VERTSCROLL
    BEGIN
        WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass"
    END
END
```

GROUPBOX Statement

The **GROUPBOX** statement creates a group-box control. The control is a rectangle that groups other controls together. A border is drawn around the groups, and text is displayed in the upper-left corner. The **GROUPBOX** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the text, identifier, coordinates, dimensions, and attributes of a group-box control. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_GROUPBOX and WS_TABSTOP.

ICON Statement (Control)

This form of the **ICON** statement creates an icon control. This control is an icon displayed in a dialog box. The **ICON** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the icon-resource identifier, icon-control identifier, position, and attributes of a control window. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_ICON. For the **ICON** statement, the width and height fields are ignored; the icon automatically sizes itself.

LISTBOX Statement

The **LISTBOX** statement creates commonly used controls for a dialog box or window. The control is a rectangle containing a list of user-selectable strings, such as file names.

The **LISTBOX** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the identifier, coordinates, dimensions, and attributes of a control window. The predefined class for this control is WC_LISTBOX. If you do not specify a style, the default style is WS_TABSTOP.

MLE Statement

The **MLE** statement creates a multiple-line entry-field control. The control is a rectangle in which the user can type and edit multiple lines of text. The control displays a pointer when the user selects it. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the Backspace and Delete keys. By using the mouse or the direction-arrow keys, the user can select the character or characters to delete or select the place to insert new characters. The **MLE** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the text, identifier, coordinates, dimensions, and attributes of a control window. The predefined class for this control is WC_MLE. If you do not specify a style, the default style is MLS_BORDER, WS_GROUP, and WS_TABSTOP. If the MLS_READONLY style is not specified, the user can edit the text.

NOTEBOOK Statement

The **NOTEBOOK** statement creates a notebook control within the dialog window. This control is used to organize information on individual pages so that it can be located and displayed easily. The **NOTEBOOK** statement defines the identifier, coordinates, dimensions, and attributes of a notebook control. The predefined class for this control is WC_NOTEBOOK. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE. The **NOTEBOOK** statement is used only in a **DIALOG** or **WINDOW** statement.

Example

This example creates a notebook control at position (20, 20) within the dialog window. The notebook has a width of 200 character units and a height of 32 character units. Its resource ID is 201. The tabs style BKS_ROUNDEDTABS specification overrides the notebook default style of square tabs. The default styles WS_TABSTOP and WS_GROUP are both in effect, though only the latter is specified.

```
#define    IDC_NOTEBOOK     201
#define    IDD_NOTEBOOKDLG  503
DIALOG "Notebook", IDD_NOTEBOOKDLG, 11, 11, 420, 420, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    NOTEBOOK   IDC_NOTEBOOK, 20, 20, 200, 400, BKS_ROUNDEDTABS | WS_GROUP
  END
```

PUSHBUTTON Statement

The **PUSHBUTTON** statement creates a push button control. The control is a round-cornered rectangle containing the given text. The control sends a message to its parent whenever the user chooses the control. The **PUSHBUTTON** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the text, identifier, coordinates, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_PUSHBUTTON and WS_TABSTOP.

RADIOBUTTON Statement

The **RADIOBUTTON** statement creates a radio-button control. The control is a small circle that has the given text displayed to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control removes the highlight and sends a message when the button is next selected. The **RADIOBUTTON** statement, which you can use only in a **DIALOG** or **WINDOW** statement, defines the text, identifier, coordinates, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_RADIOBUTTON.

SLIDER Statement

The **SLIDER** statement creates a slider control within the dialog window. This control lets the user set, display, or modify a value by moving a slider arm along a linear slider shaft. The **SLIDER** statement defines the identifier, coordinates, dimensions, and attributes of a slider control. The predefined class for this control is WC_SLIDER. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE. The **SLIDER** statement is used only in a **DIALOG** or **WINDOW** statement.

SPINBUTTON Statement

The **SPINBUTTON** statement creates a spin button control within the dialog window. This control gives the user quick access to a finite set of data. The **SPINBUTTON** statement defines the identifier, coordinates, dimensions, and attributes of a spin button control. The predefined class for this control is WC_SPINBUTTON. If you do not specify a style, the default style is WS_TABSTOP, WS_VISIBLE, and SPBS_MASTER. The **SPINBUTTON** statement is used only in a **DIALOG** or **WINDOW** statement.

Static Text Statements: LTEXT, CTEXT and RTEXT

Each of these statements creates a static text control. The control is a simple rectangle displaying the given text, which is either aligned to one edge or centered in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. These three statements, which you can use only in a **DIALOG** or **WINDOW** statement, define the text, identifier, coordinates, dimensions, and attributes of a static text control. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, WS_GROUP, and the appropriate alignment flag (DT_LEFT, DT_CENTER, or DT_RIGHT respectively).

VALUESET Statement

The **VALUESET** statement creates a value-set control within the dialog window. This control lets a user select one choice from a group of mutually exclusive choices. The **VALUESET** statement defines the identifier, coordinates, dimensions, and attributes of a value-set control. The predefined class for this control is WC_VALUESET. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE. The **VALUESET** statement is used only in a **DIALOG** or **WINDOW** statement.

Example

This example creates a value-set control at position (40, 40) within the dialog window. The value set control has a width of 220 character and a height of 20 character units. Its resource ID is 302. The style specification VS_ICON creates a control to show items in icon form. The default styles WS_TABSTOP and WS_VISIBLE are both in effect, though only WS_TABSTOP is specified.

```
#define    IDC_VALUESET     302
#define    IDD_VALUESETDLG  501
DIALOG "Value set", IDD_VALUESETDLG, 11, 11, 260, 240, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    VALUESET  IDC_VALUESET, 40, 40, 220, 160, VS_ICON | WS_TABSTOP
  END
```

A dialog template can be in either of the following:

- A resource.res file (generated from the .DLG file by the Resource Compiler)
- A block of memory that has the DLGTEMPLATE data structure, in which case you use WinCreateDlg to create the dialog box from the template.

The dialog template uses device-independent *dialog units* for the coordinate system that define the layout of controls in the dialog box.

A dialog unit is expressed in terms of the *default standard character size* , which can vary from device to device. You do not need to put code in your application to reformat the dialog box when displaying it on different devices. (Dialogs might need editing if a different system font is loaded.) A horizontal dialog unit is 0.25 of the standard character size. A vertical dialog unit is 0.125 of the standard character size. Dialog units are expressed as offsets from the origin (lower-left corner) of the dialog box.

A dialog template is a general structure. It could be termed a window template, because you can use it to define any window in an application. If you prefer, use the statement WINDOWTEMPLATE instead of DLGTEMPLATE, because it is functionally identical. This could reduce the initialization phase of the application to registering the application window classes and calling WinLoadDlg to load the template.

If you use the Dialog Editor to define a standard window, you will have to edit the resulting .DLG file to ensure that you have a client window and the required parent-child relationships. You will also have to use WinLoadMenu in your application, to create a menu bar for the window, because you cannot create menus using the Dialog Editor.

The .RES file is an object-format compiled version of the .DLG file, created when the Dialog Editor compiles the dialogs. The Dialog Editor uses the .RES file as input on any subsequent edit of the same dialog. This means that, if you use a text editor to fine-tune a .DLG file, and you want subsequently to re-edit the dialog using the Dialog Editor, you must first use the Resource Compiler to generate a new .RES file from the .DLG file.

Your application can use either the .RES file output by the Dialog Editor or a .RES file created from the .DLG file and the other resources. If your application uses the .DLG file, it must be included by the resource script file of your application.

The rcinclude statement includes the .DLG file created by the Dialog Editor; for example:

```
rcinclude dbe.dlg /* Includes .DLG file */
```

The corresponding .H file created by the Dialog Editor must also be included in the .RC file.

Using OS/2-defined control windows, OS/2 draws and operates the controls specified in the resource file for your application. Controls are windows and can be used within any other window.

Sample Dialog Template File

The following dialog template is used for the dialog described in Example.

```
    DLGINCLUDE 1 "DBE.H"


    DLGTEMPLATE mydialog LOADONCALL MOVEABLE DISCARDABLE

    BEGIN

         DIALOG "Sample Dialog Box", mydialog, 11, 8, 170, 105,
             FS_NOBYTEALIGN | FS_DLGBORDER | WS_VISIBLE |
             WS_SAVEBITS, FCF_TITLEBAR
```

```
        BEGIN

             CONTROL "Student Level:", id_null, -1, 94, 63, 9, WC_STATIC,
                 SS_TEXT | DT_LEFT | DT_TOP | WS_GROUP | WS_VISIBLE

             CONTROL "Elementary", 258, 7, 82, 62, 11, WC_BUTTON,
                 BS_RADIOBUTTON | WS_GROUP | WS_TABSTOP | WS_VISIBLE

             CONTROL "Intermediate", 259, 7, 67, 73, 9, WC_BUTTON,
                 BS_RADIOBUTTON | WS_VISIBLE

             CONTROL "Advanced", 260, 7, 51, 52, 13, WC_BUTTON,
                 BS_RADIOBUTTON | WS_VISIBLE

             CONTROL "Media", 261, 87, 48, 75, 54, WC_STATIC,
                 SS_GROUPBOX | WS_GROUP | WS_VISIBLE

             CONTROL "Textbooks", 262, 97, 83, 60, 10, WC_BUTTON,
                 BS_CHECKBOX | WS_TABSTOP | WS_VISIBLE

             CONTROL "Video", 263, 97, 68, 46, 10, WC_BUTTON,
                 BS_CHECKBOX | WS_VISIBLE

             CONTROL "CBT", 264, 97, 53, 32, 10, WC_BUTTON,
                 BS_CHECKBOX | WS_VISIBLE

             CONTROL "OK", 265, 7, 20, 38, 12, WC_BUTTON,
                 BS_PUSHBUTTON | WS_GROUP | WS_TABSTOP | WS_VISIBLE

             CONTROL "Cancel", 266, 61, 20, 38, 12, WC_BUTTON,
                 BS_PUSHBUTTON | WS_TABSTOP | WS_VISIBLE

             CONTROL "Help", 267, 117, 20, 38, 12, WC_BUTTON,
                 BS_PUSHBUTTON | WS_TABSTOP | WS_VISIBLE

        END

    END
```

Executable File Header Utility (EXEHDR)

The Executable File Header Utility (EXEHDR) displays and modifies the contents of an executable-file header. EXEHDR generates an Output listing showing the contents of the file header and information about each object or segment in the file. Options are provided that let you change values in the file header.

Uses of EXEHDR include:

- Determining whether a file is an application or a dynamic link library
- Viewing and changing the attributes set by the module definition file
- Viewing the number and size of code and data segments.

You can use EXEHDR with DOS or OS/2 applications and dynamic-link libraries.

Syntax

```
EXEHDR [options] filename
```

<options>

Options used to modify Output or change the file header.

<filename>
One or more names of applications or dynamic-link library files.

Regardless of options, EXEHDR always creates an Output listing of the file header.

Help

To display EXEHDR help, type `EXEHDR /?` at the command prompt. The appropriate copyright statement appears along with a brief list of EXEHDR options.

```
Usage: EXEHDR [options] filename...
Valid options are:
  /?
  /HEAP:(0H - ffffH)
  /HELP
  /MAX:(0H - ffffH)
  /MIN:(0H - ffffH)
  /NEWFILES
  /NOLOGO
  /PMTYPE:(PM | VIO | NOVIO | WINDOWAPI |
                WINDOWCOMPAT | NOTWINDOWCOMPAT)
  /RESETERROR
  /STACKDOS:(0H - ffffH)
  /STACKOS2:(0H - ffffffffH)
  /VERBOSE
```

Note, that for OS/2 16-bit NE modules the maximum value for /STACKOS2 is FFFFh.

Options

Usage Notes:

- Option characters are not case sensitive: /R and /r are equivalent.
- Options can be shortened to the fewest characters that uniquely identify them. The characters in brackets can be omitted: /N and /NOLOGO are equivalent.
- Although use of the minimum one-letter abbreviations is allowed, if a future release has an additional option starting with the same letter, the one-letter option will no longer be usable.
- The option /STACK is deprecated, and has been replaced by /STACKDOS and /STACKOS2. Because /STACK has serious limitations and can potentially damage an executable file, it is strongly recommended that you do not use it. Use one or both of the replacement options instead.

Formats Affected by Options

The EXEHDR options that can change executable files are MIN, MAX, HIGHMEM, NOHIGHMEM, STACKDOS, STACKOS2, PMTYPE, HEAP, RESETERROR, and NEWFILES.

Executable headers are used by the operating system to determine characteristics of the executable file, such as stack size, entry point, number of objects (or segments), and so on. EXEHDR recognizes three different kinds of executable headers: DOS (generated by DOS linker), OS/2 16-bit (generated by LINK), and OS/2 32-bit (generated by LINK386).

An X in the following table indicates which option changes which executable header:

Option	DOS	OS/2 16-bit (LINK)	OS/2 32-bit (LINK386)
HEAP		X	X
HIGHMEM			X
NOHIGHMEM			X
MAX	X		
MIN	X		
NEWFILES		X	
PMTYPE		X	X
RESETERROR		X	X
STACKDOS	X		
STACKOS2		X	X

For compatibility purposes, executable files generated by either of the OS/2 link utilities include both a DOS header and an OS/2 header.

/HEA[P]

Set Heap Allocation (/HEAP)

Syntax: `/HEA[P]:nnnn`

This option sets the size of the local heap and is applicable to OS/2 applications only. The field <nnnn> contains the local heap size in bytes.

You can specify <nnnn> in decimal, octal, or hexadecimal radix using standard C language notation.

/HI[GHMEM]

Set High Memory Flag (/HIGHMEM)

Syntax: `/HIGHMEM[:nnnn[,nnnn]]`

This option allows the specified 32-bit memory objects to be loaded into high memory (above the 512MB boundary) by the operating system loader, and is applicable to OS/2 DLL modules only.

/HIGHMEM turns on the high memory flag for each 32-bit memory object given by <nnnn>, or for all eligible 32-bit objects if no object numbers were specified. Any explicitly-specified object numbers must exist in the target module or an error message is printed.

Only 32-bit objects that don't have the ALIAS, CONFORMING, or IOPL attributes are eligible to be marked with the HIGHMEM attribute. EXEHDR ignores any request to mark ineligible objects.

You can specify <nnnn> in decimal, octal, or hexadecimal radix using standard C language notation. Memory object numbering begins at ordinal number 1. Use the output of EXEHDR to determine the valid object numbers contained in the target module.

/HEL[P] and /?

Display Help (/HELP or /?)

Syntax: `/HEL[P]` OR `/?`

This option displays a brief summary of EXEHDR syntax.

/MA[X]

Set Maximum Allocation (/MAX)

Syntax: `/MA[X]:nnnn`

This option sets the maximum allocation of memory for the program. The field <nnnn> contains the maximum number of 16-byte paragraphs required to load and run the program. This value must be equal to or greater than the minimum allocation.

Compare to /MI[N].

The Maximum Allocation option is equivalent to the LINK386 /CParmaxalloc option.

Refer to Set Max Allocation Space (/CP) (in LINK386 Help)

You can specify <nnnn> in decimal, octal, or hexadecimal radix using standard C language notation.

/MI[N]

Set Minimum Allocation (/MIN)

Syntax: `/MI[N]:nnnn`

This option sets the minimum allocation of memory for the program. The field <nnnn> contains the minimum number of 16-byte paragraphs required to load and run the program. This value must be equal to or less than the maximum allocation.

Compare to /MA[X].

You can specify <nnnn> in decimal, octal, or hexadecimal radix using standard C language notation.

/NE[WFILES]

New Files (/NEWFILES)

Syntax: `/NE[WFILES]`

This option enables long file name support for OS/2 16-bit LINK files. OS/2 32-bit LINK386 files have long file name support.

/NOH[IGHMEM]

Reset High Memory Flag (/NOHIGHMEM)

Syntax: `/NOHIGHMEM[:nnnn[,nnnn]]`

This option prevents the specified 32-bit memory objects from being loaded into high memory (above the 512MB boundary) by the operating

system loader, and is applicable to OS/2 DLL modules only.

/NOHIGHMEM turns off the high memory flag for each 32-bit memory object given by <nnnn>, or for all eligible 32-bit objects if no object numbers were specified. Any explicitly-specified object numbers must exist in the target module or an error message is printed.

You can specify <nnnn> in decimal, octal, or hexadecimal radix using standard C language notation. Memory object numbering begins at ordinal number 1. Use the output of EXEHDR to determine the valid object numbers contained in the target module.

/NO[LOGO]

Suppress Sign-On Banner (/NOLOGO)

Syntax: `/NO[LOGO]`

This option suppresses the sign-on banner displayed by EXEHDR when it starts.

/P[MTYPE]

Set Application Type (/PMTYPE)

Syntax: `/P[MTYPE]:type`

This option specifies the type of application. It pertains only to OS/2 applications. The /PMTYPE option in EXEHDR is equivalent to either the NAME Statement in the module-definition file or the Name Application Type (/PM) in LINK386.

A keyword in <type> is equivalent to a keyword in a NAME statement, as shown in the following list:

Field Keyword	Equiv. Keyword
PM	WINDOWAPI
VIO	WINDOWCOMPAT
NOVIO	NOTWINDOWCOMPAT

The NAME statement keyword is also accepted.

/R[ESETERROR]

Reset LINK386 Error (/RESETERROR)

Syntax: `/R[ESETERR0R]`

This option clears an error flag stored in OS/2 applications. The error flag is set by LINK386 when the link has unresolved external references or duplicate symbol definitions (any LINK386 error messages starting with L2xxx).

OS/2 does not load the application if the error flag is set. This option allows you to attempt to run a program with LINK386 errors and is useful during application development.

/STACKDOS

Set DOS Stack Allocation (/STACKDOS)

Syntax: /STACKDOS:*nnnn*

This option sets the size of the stack in the DOS header. The field *nnnn* contains the stack size in bytes. This option is equivalent to the Control Stack Size (/ST) (in LINK). DOS modules have a maximum stack size of 65,535.

Almost all OS/2 applications have a DOS header and DOS stub. The DOS stub normally is found in 3 forms. The first stub type is the LINK386 default stub. This is a small DOS program that simply prints an error message of the form "This program cannot be run in a DOS session." and exits. The second common stub type is a FAPI (Family API) program. This allow the applications to be run equally from a real DOS machine, an OS/2 or Windows DOS box, or from an OS/2 protected mode session. Many of the OS/2 system utilities are written with FAPI. The third stub type is that of a fully independant DOS application preceding the OS/2 binary image, allowing the two programs to share the same executable module. These DOS stubs are attached to the OS/2 module by using the STUB statement within a LINK386 .def file.

You can specify *nnnn* in decimal, hexadecimal, or octal radix using standard C language notation. For instance, a DOS stack size of 4,096 bytes can be expressed as any of the following:

- /STACKDOS:4096
- /STACKDOS:0x1000
- /STACKDOS:010000

Compare to /STACKOS2.

DO NOT USE /STACK

There is an old EXEHDR option /STACK that performed a function similar to /STACKDOS and /STACKOS2 combined. The /STACK option has serious limitations and has the potential to corrupt the header of an executable module. It it very strongly recommended that you **do not use /STACK**, and begin using /STACKDOS and /STACKOS2 instead.

- For a **DOS-only** application, the /STACK option will work correctly.
- For a 32-bit OS/2 application, the /STACK option will only change the stack value in the header of the DOS stub; it will **not** modify the OS/2 header, which is what the user may have intended.
- For a 16-bit OS/2 application, the /STACK option will attempt to change both the DOS stub and OS/2 application stack values to the same *nnnn* value. This can lead to application failures. The headers of the DOS stub and the OS/2 application should therefore be modified independently of each other using the /STACKDOS and /STACKOS2 options.

/STACKOS2

Set OS/2 Stack Allocation (/STACKOS2)

Syntax: /STACKOS2:*nnnn*

This option sets the size of the stack in the OS/2 header. The field *nnnn* contains the stack size in bytes. This option is equivalent to the Control Stack Size (/ST) (in LINK386 Help). OS/2 16-bit NE modules have a maximum stack size of 65,535. OS/2 32-bit LX modules have a maximum stack size of 4,294,967,295. The /STACKOS2 only changes the OS/2 header and has no effect on the DOS stub in the same module.

You can specify *nnnn* in decimal, hexadecimal, or octal radix using standard C language notation. For instance, an OS/2 stack size of 32,768 bytes can be expressed as any of the following:

- /STACKOS2:32768
- /STACKOS2:0x8000
- /STACKOS2:0100000

Compare to /STACKDOS.

DO NOT USE /STACK

There is an old EXEHDR option /STACK that performed a function similar to /STACKDOS and /STACKOS2 combined. The /STACK option

has serious limitations and has the potential to corrupt the header of an executable module. It it very strongly recommended that you **do not use /STACK**, and begin using /STACKDOS and /STACKOS2 instead.

- For a **DOS-only** application, the /STACK option will work correctly.
- For a 32-bit OS/2 application, the /STACK option will only change the stack value in the header of the DOS stub; it will **not** modify the OS/2 header, which is what the user may have intended.
- For a 16-bit OS/2 application, the /STACK option will attempt to change both the DOS stub and OS/2 application stack values to the same *nnnn* value. This can lead to application failures. The headers of the DOS stub and the OS/2 application should therefore be modified independently of each other using the /STACKDOS and /STACKOS2 options.

/V[ERBOSE]

Display in Verbose Mode (/VERBOSE)

Syntax: /V[ERB0SE]

This option displays the executable-file header in verbose mode.

Output

EXEHDR lists the current contents of the file header and information about each object (or segment) in the file. To redirect this output to a printer or disk file, use the operating system redirection operator.

The output is in two parts: a Header Listing giving the contents of the file header; and an Object or Segment Listing giving attributes of all objects (or segments) in the file. If the /VERBOSE option is specified, additional output is generated.

Header Listing

The header listing is comprised of the following fields:

<Module> Name of Application

This field lists the name of the application as specified in the NAME statement of the module-definition file.

If no module definition was used to create the executable file, this field displays the name assumed by default.

If a module definition was used to create the file, but the LIBRARY statement appeared instead of the NAME statement (thus specifying a dynamic-link library), the name of the library is given and EXEHDR uses the word "Library" instead of "Module" to identify the field.

<Description> Description of Application

This field gives the contents, if any, of the DESCRIPTION statement of the module-definition file used to create the file being examined.

<Data> Type of Automatic Data Object

This field indicates the type of automatic data segment in a program: SHARED, NONSHARED, or NONE. This type can be specified in a module-definition file. The defaults are NONSHARED for applications and SHARED for dynamic-link libraries.

<Initial CS:IP> Program Starting Address

This field gives the program starting address (if an application is being examined) or address of the initialization routine (if a dynamic-link library is being examined).

<Initial SS:SP> Initial Stack Pointer

This field gives the value of the initial stack pointer.

<Extra Stack Allocation> Additional stack allocation

This field gives the value of the extra stack location.

<DGROUP> Automatic-Data-Object Number

Object or Segment Listing

The object listing is comprised of the following fields:

no.	Object index number, starting with 1, in decimal
type	Identification of the object as a code or data object
	A code object is comprised of segments with class name ending in CODE. All other objects are data objects.
address	Location, within the file, of the contents of the object (in hexadecimal)
file	Size of the object (in bytes), as contained in the file (in hexadecimal)
mem	Size of the object (in bytes), as it is stored in memory (in hexadecimal)
	If the value of this field is greater than the value of <file>, the operating system pads the additional space with zero values at load time.
flags	Object attributes
	If the /VERBOSE option is not used, only non-default attributes are listed. Attributes are given in the form specified in the module-definition file.

Output Example

The following output is generated by EXEHDR for the executable file LINK386.EXE:

```
Module:                     LINK386
Description:                Operating System/2 32-bit LX Linker
Data:                       NONSHARED
Initial CS:IP:              seg   2 offset 6c78
Initial SS:SP:              seg   4 offset 0000
Extra stack allocation:     4000 bytes
DGROUP:                     seg   4

no. type address  file  mem   flags
  1 CODE 00006000 0f7d6 0f7d7
  2 CODE 00015a00 08e40 08e40
  3 DATA 0001ea00 02865 02865
  4 DATA 00021400 02337 08bd0
```

Verbose Output

When you specify the /VERBOSE option, EXEHDR generates additional output:

- DOS-specific header information. All OS/2 executable files have a DOS header, whether bound or not. If the program is not bound, the DOS portion typically consists of a stub that simply terminates the program.
- OS/2-specific header information. The object-table display in verbose mode is described below.
- File addresses and lengths of the various tables in the executable file. For each table, the following is generated:
    - Name of the table
    - Address of the table within the file
    - Length of the table in hexadecimal radix
    - Length of the table in decimal radix
- Object table with complete attributes, not just the non-default attributes. The /VERBOSE option displays two additional attributes:
    - The RELOCS attribute is displayed for each object that has address relocations. Relocations occur in each object that references objects in other objects or makes dynamic-link references.
    - The ITERATED attribute is displayed for each object that has iterated data. Iterated data consist of a special code that packs repeated bytes.
- Run-time relocations and fixups.
- All exported entry points.

Error messages

EXEHDR error messages:

EXH1100: invalid magic number *xxxx* H

{#define ER_badmagic}
EXEHDR discovered an unknown signature *xxxx* in the header for the file. The signature in the header of a file allows the operating system to identify the type of the executable module. EXEHDR only recognizes signatures for DOS modules (5A4DH), 16-bit OS/2 and Windows modules (454EH), and 32-bit OS/2 modules (584CH). Make sure the file is not a DOS .com image, a 32-bit Windows image, or other unrecognized module format.

EXH1101: automatic data segment greater than 64K; correcting heap size

{#define ER_autodata1}
There was not enough space in the automatic data segment to accommodate the requested new heap size. The heap size has been adjusted to the maximum available space. This error only occurs for 16-bit OS/2 applications. If /HEAP or /STACKOS2 is used then the total size of DGROUP plus the heap plus the stack must be less than 64K. The heap will be assigned 64K-1-DGROUP-STACK. EXEHDR issues this warning to indicate that it performed the .EXE file modification with a reduced HEAP value. Make sure the reduced value is acceptable.

EXH1102: automatic data segment greater than 64K; correcting stack size

{#define ER_autodata2}
There was not enough space in the automatic data segment to accommodate the requested new stack size. The stack size has been adjusted to the maximum available space. This error only occurs for 16-bit OS/2 applications. If /HEAP or /STACKOS2 is used then the total size of DGROUP plus the heap plus the stack must be less than 64K. This error occurs when the heap is zero. The stack will be assigned 64K-1-DGROUP. ACTION: EXEHDR issues this warning to indicate that it performed the .EXE file modification with a reduced STACK value. Make sure the reduced value is acceptable.

EXH1103: invalid .EXE file : actual length less than reported

{#define ER_badsize}
The second and third fields in the input DOS file header indicate a file size greater than the actual size of the file. This error occurs in DOS files only. EXEHDR assumes the file has been corrupted and will not perform any modifications.

EXH1104: cannot change load-high program

{#define ER_high}

When the minimum allocation value and the maximum allocation value are both 0, the file cannot be modified. Both minimum and maximum allocation of 0 is not a legal value for a DOS header. This is a DOS header only error condition. Change either /MIN or /MAX to a non-zero value.

EXH1105: minimum allocation less than stack; correcting minimum

{#define ER_minalloc1}

If the minimum allocation is not enough to accommodate the stack (either the original stack request or the modified request), the minimum allocation value is adjusted. This error applies only to DOS programs. This is a warning to indicate that EXEHDR has modified the /MIN value to create a legal DOS .EXE header. Make sure the new /MIN value is acceptable.

EXH1106: minimum allocation greater than maximum; correcting maximum

{#define ER_minalloc2}

If the minimum allocation is greater than the maximum allocation, the maximum allocation value is adjusted. If a display of DOS header values is requested, the values shown will be the values after the packed file is expanded. This error applies only to DOS programs. This is a warning to indicate that EXEHDR has modified the /MAX value to create a legal DOS .EXE header. Make sure the new /MAX value is acceptable.

EXH1107: unexpected end of resident/nonresident name table

{#define ER_minalloc2}

While decoding run-time relocation records, EXEHDR found the end of the resident/nonresident name table. The .EXE file is probably corrupted. This error applies only to OS/2 and Windows programs. The current version of EXEHDR will no longer issue this error. Therefore, report to IBM if you receive this error with the current version of EXEHDR.

EXH1108: unknown format of relocation records

{#define ER_badreloc}

EXEHDR cannot decode the information in the file header because the header is not in a standard format. The fixup relocation count is 0, which is not permitted. This error applies only to OS/2 and Windows programs.

EXH1109: illegal value 'xxxx'

{#define ER_illval}

A command-line argument to EXEHDR contained an illegal value. Retry the EXEHDR command with a correct option.

EXH1110: malformed number xxxx

{#define ER_badnum}

A command-line option for EXEHDR required a value, but the specified number was mistyped. Retry the EXEHDR command with a correct option.

EXH1111: option requires value

{#define ER_noval}

A command-line option for EXEHDR required a value, but no value was specified, or the specified value was in an illegal format for the given option. Retry the EXEHDR command with a correct option.

EXH1112: value out of legal range xxxx - xxxx

{#define ER_range1}

A command-line option for EXEHDR required a value, but the specified number did not fall in the required decimal range. The current version of EXEHDR will no longer issue this error. Therefore, report to IBM if you receive this error with the current version of EXEHDR.

EXH1113: value out of legal range xxxxH - xxxxH

{#define ER_range2}

A command-line option for EXEHDR required a value, but the specified number did not fall in the required hexadecimal range. Retry the EXEHDR command with a correct option.

EXH1114: missing option value; option xxxx ignored

{#define ER_noval1}

A command-line option for EXEHDR required a value, but nothing was specified. EXEHDR ignored the option. Retry the EXEHDR command with a correct option.

EXH1115: option xxxx ignored

{#define ER_igno}

A command-line option for EXEHDR was ignored. This error usually occurs with error U1116, unrecognized option. Retry the EXEHDR command with a correct option.

EXH1116: unrecognized option: xxxx

{#define ER_unrec}

A command-line option for EXEHDR was not recognized. This error usually occurs with either

U1115, option ignored, or U1111, option requires value. Retry the EXEHDR command with a correct option.

EXH1117: The same option has been used more than once

{#define ER_twice}
You are not permitted to enter the same option more than once. Correct the command line by entering each option only once.

EXH1118: Invalid option combination xxxx

{#define ER_combine}
Certain options can not be used together. For instance, /STACK is deprecated and replace by /STACKDOS and /STACKOS2. You are not permitted to use /STACK in combination with the replacement options. Remove the offending option.

EXH1119: /STACK deprecated, use /STACKDOS and/or /STACKOS2

{#define ER_stackdep}
The /STACK option should no longer be used. Use the /STACKDOS and/or /STACKOS2 options instead. /STACK does not work on the LX 32-bit OS/2 binary files. Furthermore, on NE 16-bit OS/2 and Windows binary files it will attempt to change both the MZ DOS stub and the 16-bit NE stack values. This is potentially dangerous because the 2 stacks may have no relationship to each other. By changing one, you could accidently damage the other. To distinguish which stack you want to change use /STACKDOS or /STACKOS2. If you need to change both stacks, then use both the /STACKDOS and /STACKOS2 options. Remove the /STACK option from your command line. Use the /STACKDOS and/or /STACKOS2 options instead.

EXH1120: input file missing

{#define ER_noinp}
No input file was specified on the EXEHDR command line. Retry the EXEHDR command with a correct filename.

EXH1121: command line too long: xxxx

{#define ER_cmdmax}
The current version of EXEHDR will no longer issue this error. Therefore, report to IBM if you receive this error with the current version of EXEHDR.

EXH1122: input filename too long: xxxx

{#define ER_filmax}
The specified filename is longer than the allowable filename size. The current maximum size is 256 characters. Retry the EXEHDR command with a correct filename.

EXH1123: object number xxxx does not exist

{#define ER_badobjnum}
An object number was specified in a /HIGHMEM or /NOHIGHMEM option that does not exist in the executable module. No adjustments are made to any HIGHMEM flags in the executable module when this condition is encountered. Run EXEHDR without the /HIGHMEM or /NOHIGHMEM options to obtain a list of the valid object numbers contained within the module. Use this information to ensure the /HIGHMEM or /NOHIGHMEM setting is being applied to the correct object number(s).

EXH1124: Invalid information level requested

{#define ER_badinfo}
DLL call to FileVerParseModuleVersion returns rc=124.

EXH1130: cannot read 'xxxx'

{#define ER_read}
EXEHDR could not read the input file. Either the file is missing or the file attribute is set to prevent reading. Make sure the file was correctly specified. If EXEHDR file modifying options are specified, make sure the file is read/write accessable. File modifying options require the ability to write to the file.

EXH1131: not valid .EXE file

{#define ER_illexe}
The input file specified on the EXEHDR command line was not a valid .EXE file.

EXH1132: unexpected end-of-file

{#define ER_eof}
EXEHDR found an unexpected end-of-file condition while reading the .EXE file. The .EXE file is probably corrupt.

EXH1133: no Import Procedure Names Table

{#define ER_noimptab}
An offset value in the executable header indicated the presence of an Import Procedure

Names Table, but additional calculations determined that the length of the table was zero; the table is therefore non-existent. The file is either corrupt, or was incorrectly generated by the linker. Regenerate the module or contact the supplier of the module to obtain a repaired version.

EXH1134: loop in internal fixup chain

{#define ER_chaincycle}
The file is either corrupt, or was incorrectly generated by the linker. Regenerate the module or contact the supplier of the module to obtain a repaired version.

EXH1140: out of memory

{#define ER_memovf}
There was not enough memory for EXEHDR to decode the header of the executable file.

EXH1150: Not a valid module version value

{#define ER_badmodver}
The "Module Version" field of the module header has not been established. Use the MARKEXE tool with the SETVERSION option to establish an appropriate value for the Module Version field.

Font Editor

You can use the OS/2 Font Editor to design and save your own fonts for use in applications.

A *font* is a set of alphanumeric characters, punctuation marks, and other symbols that share a common typeface design and line weight. An application loads a font from a dynamic-link library file (.DLL file).

The Font Editor allows you to edit an enlarged version of each character in an editing window, using the mouse to switch the enlarged representation of pels to black or white.

You can change a series of pels by dragging the mouse pointer through them while holding down the mouse button. An enlarged scale version of the character is shown in a viewing window to the right of the edit window.

Using the Font Editor

To run the Font Editor, select **Font Editor** from the **PM Development Tools** folder.

Select one of the options in the **File** menu to open a new or existing font. The letter A appears in both the editing and viewing windows. The rest of the font appears in the character selection scroll box at the bottom of the Font Editor window.

To edit any other character in the font, select it from the character selection scroll box. The character appears in the editing and viewing windows.

Font Editing Functions

Functions for defining fonts are found on the **Header** menu.

Functions for editing character width are found on the **Width** and **Shift** menus.

Defining Fonts

Use the **Header** menu to define the typestyle that you want to create:

- Select **Naming** to specify the identification details such as the type-face name.

- Select **General** to specify spacing (fixed or proportional), type face style, line width, and type weight.
- Select **Sizes** to specify the font character dimensions.
- Select **Relations** to specify the position of characters.
- Select **Definition** to change character spacing in a proportional font.

Editing Character Width

The **Width** and **Shift** menus allow you to change the width of individual characters.

The **Width** Menu

Use the **Width** menu to alter the width of a single character. This menu is enabled only when you are editing a proportional space font. You can make a character wider or narrower by adding or deleting columns of pels from the right, the left, or both sides. You may also use the **Set Character Increment** option to set the width of a character. On-line help panels describe how to perform these functions.

The **Shift** Menu

Use the **Shift** menu to insert a one-pel-wide row or column into (or delete from) the character that you are editing. When you select shift, the pointer becomes a flat horizontal or vertical bar when inside the edit window. This enables you to position it exactly where you want the operation to take place.

To cancel a shift you have selected before execution, select **Cancel Choice**.

Font Resource Files

The Font Editor creates a file with a .FNT extension. The .FNT file is not referred to in the same resource file as other resources.

Instead, it has its own resource file that contains a single-line statement that has a similar format to the ICON, POINTER, and BITMAP statements, for example:

```
FONT  101      myfont.fnt     /* Font */
```

The FONT keyword identifies the resource type.

The resource type is followed by an integer identifier that is used by the application to identify the resource. The integer is used as a parameter to the WinCreateStdWindow call. You cannot use a symbolic name for a font.

The integer identifier can be followed by loading and memory options. Again, the example lets them default.

The last part of the statement is the file name of the resource created by the Font Editor. A full path name must be given if it is not in the current directory.

Producing a font file uses a process similar to binding resources to an .EXE file. You bind one or more .FNT files to a dummy .DLL, to produce a file containing the font or fonts. The final file should have the extension .FON.

The .FON file created by the process is installed on the system and becomes a *public* font, a font that can be used by any application in the system.

A font not installed on the system is called a *private* font. Before your application can use the font, your application must use GpiLoadFonts to load the .FON file.

Forwarded Entry Point (FWDSTAMP)

FWDSTAMP adds entry points, called *forwarders* , to a dynamic link library file (.DLL). Forwarders point to API functions or other exported code or data. They contain an import reference so that the final target address of the forwarded entry is contained in a different module. A forwarder might be called an *imported export* .

When a file has a fix-up to a forwarded entry point, the loader resolves that fix-up to the address of the entry point that the forwarder imports, by traversing the chain of forwarders until the end of the chain (a nonforwarded export) is reached. All forwarders are implicitly exported.

The imported entry point that a forwarder refers to may itself be another forwarder. The loader will process a chain of forwarders until a nonforwarder entry point is encountered.

There is no *run-time* cost to forwarders; however, there is a slight *load-time* cost as the loader resolves forwarder chains with their final addresses.

Using Forwarders

You use forwarders to combine several DLLs into one without having to relink old applications. For example, if MOUCALLS and VIOCALLS were combined into a single DLL called NEWLIB.DLL, then MOUCALLS and VIOCALLS could be replaced with special DLLs containing forwarders to NEWLIB.DLL.

Important Notes

- FWDSTAMP parses only the IMPORTS and EXPORTS section of the module definition file. FWDSTAMP does not verify the syntax of the other sections.
- When exported names already exist in the input file, their attributes are kept, such as resident or nonresident names table, and ordinal numbers. Any new conflicting attributes are ignored.
- If there is no exported name, FWDSTAMP adds the one defined by the EXPORTS statement in the module definition file.

Starting FWDSTAMP

You can start FWDSTAMP and specify all input from the command line. An example of the syntax follows:

```
FWDSTAMP [options] infile deffile outfile
```

[options]	Specifies one of the following:	
	/?	Displays FWDSTAMP help panel.
	/V	Increases the level of information FWDSTAMP should output.
infile	Specifies the name of the dynamic link library file that LINK386 created. Use the file-name extension of DLL.	
deffile	Specifies the name of the module definition file (.DEF) that contains the forwarders. (See Example).	
outfile	Specifies the name of the .DLL file that will contain the added forwarders.	

Example

Forwarders are specified in the module definition file so that an exported name, which is also imported, is a forwarder. For example:

```
IMPORTS
    VIOMODEWAIT=NEWLIB.123
```

```
EXPORTS
    VIOMODEWAIT @ 25
```

In the example, a forwarder entry point for VIOMODEWAIT is created and contains an import reference to NEWLIB.123.

Generate Message Catalog Utility (GENCAT)

One of the requirements for internationalization of programs is that messages be displayed in the language of the user. This requirement is satisfied by producing versions of the messages which are translated into all supported languages.

The OS2 C library messaging support is based on the messaging support described in the X/Open XPG4 specification. It consists of functions for extracting messages from message catalog files (catopen, catgets, catclose) and utilities for producing message catalog files (gencat, mkcatdef). The functions are documented in the *C Library Reference* and the utilities are documented in this book.

The Generate Message Catalog Utility (GENCAT) creates and modifies a message catalog. GENCAT creates the message catalog (usually *.cat) from a message text source file (usually *.msg) or standard input.

If a message catalog with the name specified by the *CatalogFile* parameter exists, GENCAT modifies it according to the statements in the specified message source files. If the message catalog does not exist, GENCAT creates a catalog file with the name specified by the *CatalogFile* parameter.

You can specify any number of message source files. GENCAT processes multiple source files, one after another, in the sequence specified. Each successive source file modifies the catalog. If you do not specify a source file, GENCAT accepts message source data from standard input.

GENCAT does not accept symbolic message identifiers. You must run MKCATDEF if you want to use symbolic message identifiers in your message source file (see Preprocess Message Source File Utility (MKCATDEF)).

Syntax

```
gencat CatalogFile [ SourceFile ... ]
```

Examples

To generate a TEST.CAT catalog from the source file TEST.MSG, enter:

```
gencat test.cat test.msg
```

The TEST.MSG file does not contain symbolic identifiers.

Source File Syntax

All fields of a source line are separated by a single blank character. Any other blank characters are considered as being part of the subsequent field.

Comments begin with a $ followed by a space and the text of the comment. For example:

```
$ This is a comment
```

A $set directive is used to associate a message number with all subsequent messages, until the end of file or the next $set directive. The format is:

```
$set <number> <comment>
```

For example:

```
$set 1 this is the first set in the file
```

The set number must be in the range [1, {NL_SETMAX}]. Set numbers must be ascending within the file, but need not be contiguous. Any string following the set number is treated as a comment. If no $set directive is specified in a message text source file, all messages will be located in an implementation-defined default message set NL_SETD.

A $delset directive is used to delete a message set from an existing message file. The format is:

```
$delset <number> <comment>
```

For example:

```
$delset 1 this deletes any previous definition of set 1 from this file
```

A message consists of a message number followed by the message text. The message number must be in the range [1, {NL_MSGMAX}]. The message text is stored in the message catalog with the set identifier specified by the last $set directive, and with message the message number. If the message text is empty, and a blank character field separator is present, an empty string is stored in the message catalog. If a message source line has a message number, but neither a field separator nor message text, the existing message with that number (if any) is deleted from the set. Message numbers must be in ascending order with a single set, but need not be contiguous. The length of the message text must be in the range [0, {NL_TEXTMAX}].

For example:

```
1 "This is the first message"
2 "This is the second message"
```

A $quote directive specifies an optional quote character, which can be used to surround message text so that trailing spaces or null messages are visible in a message source line. By default, or if an empty $quote directive is supplied, no quoting of message text will be recognized. The format is:

```
$quote <character>
```

For example:

```
$quote "
```

Empty lines in the message file are ignored. The effects of lines starting with any character other than those defined above are implementation-defined.

Text strings can contain the special characters and escape sequences defined in the following table:

Description	Sequence
newline	\n
horizontal tab	\t

vertical tab	\v
backspace	\b
carriage-return	\r
form-feed	\f
backslash	\\
octal character encoding	\ddd (three or fewer octal digits)

If the character following a backslash is not one of the above or the end of line, the backslash is ignored. A backslash at the end of a line is used to continue a string on the following line. For example:

```
1 This line continues \
to the next line
```

Related Information

- MKCATDEF (see Preprocess Message Source File Utility (MKCATDEF))
- catclose, catgets and catopen (see *C Library Reference*)

Icon Editor

The Icon Editor lets you create your own art (icons, pointers, and bit maps) and save them for use by applications.

Icons, pointers, and bit maps produced by the Icon Editor are graphic symbols that are made up of pels (also known as pixels) in any of the following display states:

- Black
- White
- Color
- Screen (background color)
- Inverse screen (inverse of background color)

An application can use an icon to represent a minimized standard window. For example, an application that lists telephone numbers could use a telephone icon when minimized. An application can also use icons as warning symbols in message boxes (for example, an exclamation mark or an upraised hand).

An application can associate a pointer with the mouse or similar pointing device, so that the user can move the pointer around the screen to select controls or text. A pointer could also be used in an interactive graphics application to draw graphics on the screen. For example, a free-hand graphics-drawing application could use a pencil shape to represent the pointer.

Using the Icon Editor

To run the Icon Editor, select the **Development Tools** folder and then select **Icon Editor**.

The Icon Editor window consists of three parts: the information panel, the palette window, and the editing window.

The information panel at the top of the Icon Editor window displays the following information:

- A picture of a two-button mouse, showing the colors currently selected for each button
- An actual-size image of the current figure that you are editing
- The status area, showing the following:
    - Size (defined as 32 x 32 for icons and pointers; user-defined for bit maps)
    - Pen location
    - Pen size (from 1 x 1 to 9 x 9)
    - Hot spot (for icons and pointers, but not bit maps)
    - Figure type (icon, pointer, or bit map)
    - Form name

The palette window, in the lower-right corner, displays the colors that are available for use during editing. The colors currently selected are marked with frames.

The editing window is the largest part of your working area. Use the mouse or keyboard to move the pointer, clicking or dragging the pointer to paint the enlarged representation of pels with the selected color.

Creating a Figure

The **Edit** menu includes the functions used to select an icon, pointer, or bit map for editing, and to save it after you are through.

Selecting your icon, pointer, or bit map

- To create a new icon, pointer, or bit map, select **New** from the **File** menu. The New Figure pop-up window appears, prompting you for more information.

  Select the figure type: **Icon**, **Pointer**, or **Bit map**. For a bit map, you must specify the width and height in pels. Select Enter.

  You can also create new art by modifying or editing an existing art of the same type.

- To edit existing art, select **Open** from the **File** menu. You will be prompted for a name.

  Note: Unless you have turned off Safe Prompting on the **Options** menu, you will be prompted to save if you select **Open** or **New** while there is unsaved art on your screen.

- If you started the Icon Editor from a command prompt and specified multiple files, you can use the **Next** option on the **File** menu to select the next file.

  The **Next** option will not be selectable if you did not start from the command line and specify multiple files.

Saving your icon, pointer, or bit map

To save your art, select either of the following:

- **Save** to save it under its current file name. If this is new art, you will be prompted for a name.
- **Save As** to save it under a different name. You will be prompted for a new name.

Editing Art

To edit your art, use the functions of the **Edit** menu.

Select **Undo** to restore the art to the way it was before the most-recent editing operation.

Four of the editing functions require that you first mark the area to be edited, using **Select** or **Select All**.

If you choose **Select**, the cursor changes to a plus (+) inside a square. Hold mouse button 1 down to anchor one corner, and then drag the mouse. Release the button to anchor the opposite corner of the rectangular area you want to edit.

If you choose **Select All**, the entire figure is selected.

The following functions require that an area be selected first:

- **Fill** to fill the selected area with the current palette color. For additional information, see Filling Areas with Color.
- **Cut** to cut an area you would like to move or delete.
- **Copy** to copy an area you would like to duplicate elsewhere in the same file or in another file.
- **Paste** to place an area you have marked with **Cut** or **Copy**. Drag the outlined area that you have marked to the place you would like to paste it.
- **Clear** to erase all drawing within an area you have selected and leave transparent pels. If you have used **Select All**, this will clear your entire icon, pointer, or bit map.
- **Stretch Paste** to paste the clipboard contents into your art, stretching and positioning them to fit.
- **Flip Horizontal** to flip the art on its horizontal axis, reversing bottom and top.
- **Flip Vertical** to flip the art on its vertical axis, reversing left and right. You can create a symmetrical drawing by copying one side of the art to the other side, and then flipping one of them.
- **Circle** to inscribe a circle or ellipse within the selected area.

Using Options

The choices on the **Options** menu enable you to test your art and vary your editing environment. To change an option, from the **Options** menu, select:

Test
: To view the pointer or icon you are editing. The pointer or icon will be displayed, in actual size, as the pointer until you toggle back by again selecting **Test** from the **Options** menu.

Grid
: To superimpose a grid over the editing window. This can be useful when you want to draw a symmetrical figure. Each cell of the grid represents one pel in the figure.

X background
: To make the transparent pels (where the background is visible) apparent when editing an icon or pointer. All screen or inverse-screen colors will be shown with an X. This option does not apply to bit maps because they have no transparent pels.

Draw Straight
: To temporarily restrict your drawing to straight vertical and horizontal lines. Even if you deviate from the horizontal row, a horizontal line is produced when the mouse pointer is dragged across the editing window. Dragging the mouse up or down produces straight vertical lines.

Changing Pen Size
: Select **Pen size** on the **Options** menu to specify how many pels the pointer paints at a time. You can select any of nine square pen sizes:

1x1	4x4	7x7
2x2	5x5	8x8
3x3	6x6	9x9

Shortcut: Select a pen size by pressing Ctrl and the size, such as Ctrl+6 for a 6 x 6 pen size.

Setting Preferences
: To change your preferences, select **Preferences** from the **Options** menu. Then select any of the following:

Safe prompting
: Provide a warning before destructive operations such as file overwrites.

Suppress Warnings
: Suppress display of informational messages.

Save state on exit
: Save settings for your next session.

Display status area
: Toggle between the picture of the mouse and the art from the status area.

Reset options and modes
: Deselect the following items:

Select
Hot Spot
Color Fill
Find Color The palette will not be reset.

Changing Pen Shape

Before you select **Pen Shape**, you must first select the shape using the **Select** function on the **Edit** menu. See Editing Art for information about **Select**. Then select **Set Pen Shape** on the **Options** menu.

Defining a Hot Spot

The *Hot spot* is the pel where mouse input for an icon or pointer is directed. The default hot spot location is 16 x 16, the center of the icon or pointer. Bit maps do not have hot spots.

Select **Hot spot** from the **Options** menu to designate this pel. The cursor changes shape, and the screen coordinates of the current hot spot are displayed in the information window. When you click on a new hot spot, the screen coordinates of the new hot spot are displayed.

Select **Hotspot** again to return to editing.

When an application uses WinQueryPointerPos to query the screen position of a pointer, the OS/2 operating system returns the coordinates of the pointer hot spot.

Selecting Colors

Use **Palette** to select a new drawing color, using mouse button 1 or 2.

The currently selected color for mouse button 2 is framed on the palette in red; the color for mouse button 1 is framed in green. The currently selected colors for both mouse buttons are also displayed at the left side of the status area.

Changing Palettes or Palette Colors

To change palettes or palette colors, select the **Palette** menu. On the **Palette** menu, you can:

- Select **New** to create a new palette. The default palette will appear for you to edit.
- Select **Open** to open an existing palette.
- Select **Save** to save your current palette. If it is a new palette, you will be prompted for a name.
- Select **Save as** to save the palette under a different name. You will be prompted for a new name.
- Select **Edit color** to edit a color in your palette.
- Select **Swap colors** to swap the colors of mouse buttons 1 and 2. A submenu will appear, asking whether you want to preserve these colors in your art. Unless you select **Preserve figure**, the colors in your art will be changed accordingly.
- Select **Set default palette** to save the existing palette as your default palette.

Editing Palette Colors

To change the colors that appear on your palette, follow these steps:

1. Select the color to be edited with the mouse. A frame appears around it on the palette.
2. Select **Edit color** from the **Palette** menu.

   Shortcuts:

   - Double-click on the color to be edited.
   - To select a color that you have already used in your art, use **Find color** on the Tools menu.

   The Edit Color window will appear.
3. You can change the way you define palette colors by checking **Dynamic editing** and **Important** and choosing between **RGB** and **HSV** terms.

   - **Dynamic editing**, when checked, will make your art change dynamically as you edit individual colors, so that you can see how the changes will affect your art.
   - **Important**, when checked, will require that the color be accurately rendered, without *dithering* (approximating the color).
   - Every color can be described numerically in either RGB or HSV terms:

RGB	As proportions of the primary colors red, blue, and green
HSV	In terms of hue, saturation, and value

     To toggle between RGB and HSV, select the appropriate radio button.
4. Use the scroll bars to change RGB or HSV values, or change these numbers from the keyboard.
5. Select **OK** to save the edited color.

Filling Areas with Color

There are two ways to fill an area with color:

- To fill an irregularly-shaped area with the current palette color, select **Color fill** from the Tools menu.

  After you click on a specific pel, all adjoining areas that are the same color as that pel will be colored with the selected color.
- To fill a previously-selected area with the current palette color, select **Fill** from the **Edit** menu. You must first select an area. See Editing Art for information about **Select** and **Select All**.

Note: To select a color that you have already used in your art, use **Find color** on the **Tools** menu. A question-mark-arrow cursor will appear.

Click on a specific pel of that color, and that color is selected.

Creating Icons for Specific Displays

Although the Icon Editor edits and saves a device-independent form of the icon, the **Device** menu enables you to create versions of the icon

for specific display devices.

An independent form is automatically created when you create a new icon or pointer and all other forms are derived from it. If you select any of the other device forms listed in the menu, a new form is created for the specified device. The **Custom** option enables you to create an icon or pointer for any other device.

Select **List** to view a list of all existing forms, including custom and standard forms. Any item in this list can be selected and edited or deleted. However, you must have at least one device-independent form. Select **Add** in the list dialog to add a new device form.

Several icon bit maps can be saved in a single icon resource; when the icon is saved, all versions are saved with it in a format that includes a device resolution tag for each version. When the icon is loaded from a resource file, the display device resolution is matched against the device for which each device-dependent icon was intended. If a match is found, that icon is used. If no match is found, the application uses the device-independent icon, which always exists.

Figure files can contain any of the following forms to support multiple devices:

- Independent
- CGA (2 colors)
- EGA (16 colors)
- VGA (16 colors)
- XGA/8514 (256 colors)
- XGA/8514 (16 colors)
- XGA/8514 Small Color Form (16 colors)
- XGA/8514 Small BW Form
- Custom

Device-dependent icons are those that are designed for a particular display resolution.

An application can display icons or bit maps in dialog boxes or windows.

The file-name extension depends on the type of resource you are creating. The Icon Editor produces a file with any of the following extensions:

ICO for icons
PTR for pointers
BMP for bit maps

The ICO, PTR, or BMP files must be referred to in the resource script file for your application. The external files containing icons, pointers, and bit maps are all referred to in the resource script file by single-line statements that have a similar format. For example:

```
ICON    ID_MAINWND myprog.ico   /* Icon    */

POINTER ID_PTR     mypoint.ptr  /* Pointer */

BITMAP  ID_BMP     mybtmp.bmp   /* bit map  */
```

ICON, POINTER, and BITMAP keywords identify the resource type.

The resource type is followed by a symbolic name or integer identifier that is used by your application to identify the resource. For example, with ICON, the ID_MAINWND identifier can be used by the application in the control data parameter of the WinCreateWindow call (or as a parameter to the WinCreateStdWindow call) that creates the frame of the main window of your application. The OS/2 operating system then associates the icon with the main window.

The symbolic name or identifier can be followed by any loading and memory options.

The last part of the statement is the file name and file type of the resource created by the Icon Editor. A fully qualified path name must be given if the file is not in the current directory. An icon that it used for a minimized application should have the same file name as the executable file of the application.

Using a Command Line

If you start the Icon Editor from a command line rather than from an icon, you have an additional option available. You can load more than one file at a time by specifying the files on the command line. For example, the following command would load the two specified icons, a bit map, and a pointer:

```
ICONEDIT Ruth.ico gurp.ico alex.bmp pamela.ptr
```

If you specify multiple files when you start the Icon Editor from the command line, you can use the **Next** option on the **File** menu to select the next file. This option is available *only* if you specify multiple files from the command line.

Managing Import Libraries (IMPLIB)

IMPLIB creates import libraries used to link dynamic-link libraries with applications.

Import libraries are created by IMPLIB and used to link dynamic-link libraries with applications.

What Are Import Libraries?

Import libraries are similar in some respects to standard libraries:

- You specify import libraries and standard libraries in the same command line field of LINK386 (see Link for Object and Library Files (LINK386) for information on LINK386) or OS/2 16-bit LINK.
- Both kinds of libraries resolve external references at link time.

However, import libraries differ from standard libraries in that they contain no executable code. Rather, they identify the dynamic-link libraries where the executable code can be found at run time.

Why Use Import Libraries?

Creating import libraries is an extra step. Nevertheless, import libraries are recommended for use with all dynamic-link libraries for two reasons:

- IMPLIB automates much of the program creation process for you. To use IMPLIB, supply it with the .DEF file you already created for the dynamic-link library. Without an import library, you must create a second .DEF file that explicitly defines all needed functions in the dynamic-link library.
- Import libraries make it easier for one person to write a library and another to write the application. Much of the linking process (linking the .DLL file and creating the import library) can be done by the author of the dynamic-link library. The import library and associated .DLL file can then be given as a unit to the person linking the application - that person need not worry about creating a .DEF file.

Running IMPLIB

The following parameters can be used with IMPLIB:

<options>
: The option that modifies the IMPLIB output. See IMPLIB Options

<implibname>
: Import library created

<deffile>
: One or more module definition files that export routines in the dynamic-link library

<dllfile>
: One or more dynamic-link libraries with exported routines.

You can specify any number of either module definition files or dynamic-link libraries.

By using the _export keyword in C, you can declare functions that are exported from the dynamic-link library.

IMPLIB Syntax

Syntax

`IMPLIB [options] implibname {deffile...| dllfile...}`

Example

The following command creates the import library named MYLIB.LIB from the module definition file MYLIB.DEF.

```
IMPLIB MYLIB.LIB MYLIB.DEF
```

IMPLIB Help

To display IMPLIB help, type the following at the command prompt:

```
IMPLIB /?
```

The appropriate copyright statement is displayed along with a list of IMPLIB options.

```
Usage: IMPLIB [options] implibname {deffile... | dllfile...}
Valid options are:
    /?
    /HELP
    /IGNORECASE
    /NOIGNORECASE
    /NOLOGO
```

IMPLIB Options

Usage Notes:

- Option characters are not case sensitive: /H and /h are equivalent.
- Options can be shortened to the fewest characters that uniquely identify them. The characters in brackets can be omitted: /NOL and /NOLOGO are equivalent.
- Although use of the minimum one-letter abbreviations is allowed, if a future release has an additional option starting with the same letter, the one-letter option will no longer be usable.

The following options may be used with IMPLIB:

/?
: Displays a short summary of IMPLIB syntax.

/H[ELP]
: Displays a short summary of IMPLIB syntax.

/I[GNORECASE]
: Turns case sensitivity off. This is the default.

/I[GNORECASE]
: Turns case sensitivity on. By default, case sensitivity is off.

/NOL[OGO]
: Suppresses the sign-on banner when IMPLIB starts.

IMPLIB Error Messages

There are two types of IMPLIB error messages:

- Fatal errors cause IMPLIB to stop running. Message numbers IM1600 through IM1606 report these problems.
- Nonfatal errors indicate problems in the library file. IMPLIB produces the library file and sets the error bit in the header for the OS/2 environment. This means that the library file cannot be run from OS/2. Message numbers IM2600 through IM2604 report these problems.

IM1600

error while writing to output file - *name*
Explanation: There was not enough disk space available to create the target library.
Action: Delete or move files to make space on the disk and restart IMPLIB.

IM1601

out of memory - *heap name* heap exhausted
Explanation: There was not enough memory available to run IMPLIB.
Action: Reduce the number of programs presently running in your system and restart IMPLIB.

IM1602

error in the module definitions file
Explanation: There was an error in the module definition (.DEF) file.
Action: Correct the symbol shown in the message, at the line number given. Restart IMPLIB.

IM1603

name : cannot create file *reason*
Explanation: IMPLIB was unable to open or create the target library specified.
Action: Check the file name and available space. Restart IMPLIB.

IM1604

name : cannot open file - *reason*
Explanation: IMPLIB was unable to open one of the specified input module definition (.DEF) files.
Action: Check the file name. Restart IMPLIB.

IM1605

too many nested include files in module definition file
Explanation: The .DEF file exceeded a nesting level.
Action: Combine some nestings into one .DEF file and try again.

IM1606

missing or bad include file name
Explanation: IMPLIB could not find a file included by .DEF file.
Action: Make sure the file exists and can be located and that any path is specified correctly. Try again.

IM2600

line *number* is too long; truncated to *length* characters
Explanation: You have tried to export more than 8192 names.
Action: Retry with fewer names, creating an additional executable module if necessary.

IM2601

symbol multiply defined
Explanation: An export name was repeated within or across the module definition (.DEF) files.
Action: Eliminate duplicate definitions of the export name.

IM2602

unexpected end of name table in DLL
Explanation: IMPLIB encountered an error when reading the module names table or procedures names table.
Action: The 9per.DLL file is corrupted. Check to make sure that the DLL was generated by OS/2 LINK or LINK386.

IM2603

name : invalid .DLL file
Explanation: IMPLIB could not parse the DLL properly.
Action: The .DLL file is corrupted. Check to make sure that the DLL was generated by OS/2 LINK or LINK386.

IM2604

unrecognized option '*option*'; option ignored
Explanation: You specified an incorrect option.

Action: Specify a correct IMPLIB option.

Quick Information (KwikINF)

KwikINF provides you with a quick and convenient method of accessing information in online documents stored in the OS/2 BOOKSHELF from anywhere on the desktop, with the exception of DOS or WIN-OS/2 sessions. When KwikINF has been started, you can open a dialog with KwikINF by pressing a user-selectable hot key. Until you configure KwikINF, your KwikINF hot key is ALT+Q. The KwikINF window includes a **Configure** push button. This opens another dialog with KwikINF: the Configure KwikINF window. The KwikINF window also allows you to initiate searches for text strings in online documents of choice.

Automatic Text Retrieval

The KwikINF window includes a **Search String** entry field. You can specify the text string you want KwikINF to search for. Or, under certain conditions, this entry field automatically contains the word located under the cursor when you press your KwikINF hot key. This text retrieval feature is available from OS/2 full-screen and Presentation Manager (PM) multi-line entry (MLE) fields. This feature is also available from PM AVIO and VIO windows. Communication Manager's 3270 emulator is a common example of a PM AVIO window. An OS/2 Window is a VIO window. This means that if, for example, you open an OS/2 Window and start an OS/2 text-based application, KwikINF will automatically retrieve the word under the cursor when you press your KwikINF hot key. This automatic text-retrieval feature is not available from graphic-text PM windows.

BOOKSHELF Online Documents

The KwikINF window includes a **Volume to Search** list box of all online documents stored in the OS/2 BOOKSHELF subdirectories. KwikINF initiates searches for information in any online document in this list.

The BOOKSHELF is an environment variable, set in CONFIG.SYS, that contains a list of subdirectories containing online documents created as viewable .INF files with the Information Presentation Facility (IPF). The BOOKSHELF environment variable is set as follows:

SET BOOKSHELF=<*subdirectory*>;...;<*subdirectory*>;

Online documents for OS/2 (for example, the Command Reference) are stored in the \OS2\BOOK subdirectory of the drive on which OS/2 is installed. Online documents for the OS/2 Toolkit (for example, the Programming References) are stored in the \TOOLKIT\BOOK subdirectory of the drive specified during installation of the online programming information. As an example, after installation of OS/2 and the Toolkit, the BOOKSHELF environment variable is set as follows:

SET BOOKSHELF=C:\OS2\BOOK;D:\TOOLKIT\BOOK;

Where c: is the drive where OS/2 is installed and D: is the drive where the Toolkit is installed.

The online document where KwikINF looks for the **Search String** is selected from the **Volume to Search** list box by KwikINF or by you. KwikINF selects the **Volume to Search** by looking for the text string that has a matching entry in the KwikINF index file or, if there is no matching entry in the index file, in the **Default Volume** you have selected in the Configure KwikINF window.

Index Files for Rapid Search

The KwikINF index file provides a rapid-search mechanism for locating specific kinds of information in online documents in the BOOKSHELF. The KwikINF index file consists of one or more concatenated files stored in the BOOKSHELF and defined by the HELPNDX variable in CONFIG.SYS as shown in the following example:

```
SET HELPNDX=EPMKWHLP.NDX
```

where EPMKWHLP.NDX is the KwikINF index file for the OS/2 Toolkit.

As an example, the following excerpt from the KwikINF index file for the OS/2 Toolkit:

```
EXTENSIONS: *
DESCRIPTION: IBM Developer's Toolkit for OS/2
(IPF*, view ipfc20.inf ~)
(WinCreateWindow, view pmwin.inf ~)
```

is used by KwikINF to quickly locate **Search String** entries with the prefix IPF in the IPF-viewable file IPFC20.INF. and to quickly locate the specific **Search String** entry WinCreateWindow in the IPF-viewable file PMWIN.INF.

The first token in a rapid-search string is a specific text string (for example, WinCreateWindow) or prefix wildcard (for example, IPF*). It is also used as the text-string you want VIEW.EXE to locate within the online document. The second token is the name of the IPF file viewer (VIEW.EXE). The third token is a parameter for VIEW.EXE: the name of the .INF file that contains the online document. Currently, the fourth token is not being used and it is treated as a comment.

Enabling Online Documents

You can enable any online document for KwikINF by:

1. Creating the online document as a viewable .INF file using the Information Presentation Facility (IPF).
2. Appending the name of the subdirectory where it is stored to the BOOKSHELF in CONFIG.SYS.
3. Creating an index file to support the KwikINF rapid-search mechanism, storing it in the BOOKSHELF, and adding it to the HELPNDX variable in CONFIG.SYS.

For example, you can enable your online document MYDOC stored in MYSUBDIR subdirectory for KwikINF by:

1. Compiling the tagged source for MYDOC with the IPF compiler by entering:

   ```
   IPFC MYDOC /INF
   ```

2. Modifying the BOOKSHELF statement in CONFIG.SYS as follows:

   ```
   SET BOOKSHELF=...;C:\MYSUBDIR;
   ```

3. Creating MYINDEX file in MYSUBDIR as shown below:

   ```
   /* C style comments and blank lines are acceptable */
   /* specific file extensions may be specified here  */
   EXTENSIONS: *
   /* a title may be placed here                      */
   DESCRIPTION: Any custom KwikINF index file
   /* rapid-search strings                            */
   (thisfunction, view mydoc.inf ~)
   (my*, view mydoc.inf ~)
   ```

4. Modifying the HELPNDX variable in CONFIG.SYS as follows:

   ```
   SET HELPNDX=EPMKWHLP.NDX+MYINDEX.NDX
   ```

For more information on creating an IPF-viewable online document, see the *IPF Reference* in the Toolkit Information folder.

Using KwikINF

KwikINF is installed as a program object in the OS/2 Toolkit Information folder. You start KwikINF by double-clicking on the KwikINF object or by entering KwikINF from the command line of an OS/2 Window. You can start KwikINF automatically when you start OS/2 by placing a shadow of the KwikINF object in the Startup folder in the OS/2 System folder on the desktop. You shadow an object by pressing CTRL + SHIFT while dragging the object.

KwikINF installs a PM system hook to monitor keystrokes in PM sessions and OS/2 character device monitors to monitor keystrokes in OS/2 full-screen sessions. KwikINF will install only one copy of the hook and monitors, even if you attempt to re-start KwikINF.

When KwikINF has been started, you can initiate searches for text strings in online documents of choice by pressing a user-selectable hot key.

Note: You cannot initiate searches for text strings in online documents from DOS or WIN-OS/2 sessions.

Until you configure KwikINF, your KwikINF hot key is ALT+Q. You configure KwikINF by pressing your KwikINF hot key and then pressing the **Configure** push button to open the Configure KwikINF window.

Note: When you start KwikINF by double-clicking on the KwikINF object in the Toolkit Information folder, a message box tells you what hot key opens the KwikINF window. This technique can also be used to determine what your current KwikINF hot key is, after KwikINF has been started.

How you initiate a search for information in online documents is dependent on where you are on the desktop when you press the KwikINF hot key:

- From an OS/2 full-screen session, a PM VIO or AVIO window, or PM MLE: position the cursor on the string you want to search for and press the KwikINF hot key. KwikINF retrieves the word at the cursor. If you have configured KwikINF to display the KwikINF window when the KwikINF hot key is pressed, KwikINF automatically places the retrieved word in the **Search String** entry field of the KwikINF window. When you press the **Search** push button or Enter, KwikINF displays the information. If you have configured KwikINF to bypass the KwikINF window when the KwikINF hot key is pressed, KwikINF automatically displays the information.

  If no word is under the cursor, the previous **Search String** is used. If no previous **Search String** exists, KwikINF displays the Contents of the **Default volume to search**.

- From a graphic-text PM window: press the KwikINF hot key, then type the string you want to search for in the **Search String** entry field of the KwikINF window. The KwikINF text-retrieval feature is not available from graphic-text PM windows.

The online document where KwikINF looks for the text string is selected from the **Volume to Search** list box by KwikINF or by you. To open the online document to the panel that contains the information, press the **Search** push button or press Enter.

KwikINF From the Command Line

You can start, terminate, and configure KwikINF from the command line in an OS/2 Window by entering:

```
KwikINF [no options] [/C] [/T] [/?]
```

where:

no options	starts KwikINF. After entering this command, the default KwikINF hot key (ALT + Q) is enabled.
/T	terminates KwikINF and disables the KwikINF hot key.
/C	opens the Configure KwikINF window. Use this window to select another KwikINF hot key, to select a default online document from the BOOKSHELF to search, and to select the activation behavior of the KwikINF window.
/?	displays the following information.

```
 Usage: KwikINF [Option]
  Option    Description
    /C      Configure KwikINF
    /T      Terminate KwikINF
    /?      This short help list
```

Configuring KwikINF

You configure KwikINF through the Configure KwikINF window. KwikINF displays this window when you press the **Configure** push button on the KwikINF window or when you enter the following from the command line of an OS/2 Window:

```
KwikINF /C
```

The Configure KwikINF window allows you to:

- Select another KwikINF hot key.
- Specify the number of OS/2 full-screen sessions enabled for KwikINF.
- Specify the name of the default online document KwikINF searches.
- Select the activation behavior of the KwikINF window.

Use the push buttons on the Configure KwikINF window as follows:

- Press **OK** to enable your configuration choices.
- Press **Cancel** to cancel your configuration choices. This closes the Configure KwikINF window.
- Press **Help** to get general help for the current window.

Activation Key Sequence

The **Activation Key Sequence** provides a selectable list of KwikINF hot keys. To access the list, single-click on the down arrow. Select the KwikINF hot key of your choice from the following list:

CTRL + A

CTRL + H

CTRL + Q

ALT + A

ALT + Q (this is the default hot key)

The KwikINF hot key initiates searches for information in online documents from anywhere on the desktop, with the exception of DOS or WIN-OS/2 sessions.

Full Screen Sessions

Use the **Number of Fullscreen Sessions to Monitor** spin button to specify the number of OS/2 full-screen sessions enabled for KwikINF. KwikINF is implemented as a PM system hook to monitor keystrokes in PM sessions and as OS/2 character device monitors to monitor keystrokes in OS/2 full-screen sessions. For OS/2 full-screen sessions, KwikINF will monitor only the number of sessions specified here.

Default Volume to Search

Use the **Default Volume to Search** single selection list box to specify which online document in the BOOKSHELF you want KwikINF to search by default. KwikINF looks for the **Search String** in this online document, when there is no matching entry in the KwikINF index file. Select the online document, then select the **OK** push button to activate the selection.

Activation Behavior

Use the **Activation Behavior** radio buttons to select the behavior of the KwikINF window. The KwikINF window can be displayed or bypassed when the user presses the KwikINF hot key after KwikINF has been installed.

- Select the **Display KwikINF Window** radio button to tell KwikINF that you always want the KwikINF window to be displayed when you press the KwikINF hot key to initiate searches for information. This is the default behavior of the KwikINF window.

  When the you press the KwikINF hot key, you can initiate a search for the text string that may be automatically displayed in the **Search String** entry field or for the text string that you enter into this field. You can also specify which online document in the BOOKSHELF KwikINF searches for the text string.

- Select the **Bypass KwikINF Window** radio button to tell KwikINF that you do not want the KwikINF window to be displayed when you press the KwikINF hot key to initiate searches for information. This is typically used when working under conditions where the KwikINF automatic text-retrieval feature is available.

  When you press the KwikINF hot key, KwikINF automatically looks for the text string under the cursor in the online document that has a matching entry in the KwikINF index file or, if there is no matching entry in the index file, in the **Default volume** selected from the Configure KwikINF window.

  You configure KwikINF by pressing the **Configure** push button in the KwikINF window. To configure KwikINF when this window is bypassed, press SHIFT + your KwikINF hotkey to display the Configure KwikINF window.

Searching Using the KwikINF Window

If you have configured KwikINF to display the KwikINF window (this is the default condition), the KwikINF window is displayed when you press your KwikINF hot key.

The KwikINF window allows you to search for a text string in an online document in the OS/2 BOOKSHELF. The text string is typed by you in the **Search String** entry field or is automatically retrieved by KwikINF, under certain conditions, from under the cursor when you press your KwikINF hot key.

The online document that KwikINF searches for the text string is selected from the **Volume to Search** list box by KwikINF or by you. KwikINF selects the **Volume to Search** by looking for the text string that has a matching entry in the KwikINF index file or, if there is no matching entry in the index file, in the **Default Volume** you have selected in the Configure KwikINF window. Or you can override KwikINF's selection of the **Volume to Search** by making your own selection from the list box.

To initiate the search for the text string in the online document, press the **Search** push button or press Enter. If the search is successful, KwikINF opens the online document to the online panel that contains the information and displays a window with a title bar that matches the search string.

If the search is not successful, you can search any online document for the information by following this procedure:

- Clear the **Search String** entry field.
- Select an online document from the **Volume to Search** list box. The Contents window of the online document appears.
- Select **Services** from the menu bar.
- Select **Search** from the **Services** pull down menu. The **Search** help window appears.
- Type the text string, then select the **All libraries** radio button.
- Select the **Search** push button or press Enter.

Use the push buttons on the KwikINF window as follows:

- Press **Search** to initiate the search for the text string in the **Search String** entry field in the selected online document.
- Press **Cancel** to cancel the request to search for the text string and to close the KwikINF window.
- Press **Configure** to display the Configure KwikINF window.
- Press **Help** to get general help for the current window.

Search String Entry Field

KwikINF searches for the text string in this entry field in the selected online document in the OS/2 BOOKSHELF.

Under certain conditions, KwikINF automatically retrieves the word under the cursor when you press your KwikINF hot key. Letters, numbers, underscores, and the pound sign are retrievable by KwikINF. Blank spaces and other special characters are used as delimiters and are not retrievable by KwikINF.

You may also type any text string you want into this field. All characters are valid in the entry field to allow for special search criteria.

VOLUME TO SEARCH List Box

The KwikINF window includes a list box of all online documents stored in the OS/2 BOOKSHELF subdirectories. KwikINF initiates searches for information in any online document in this list. The online document where KwikINF looks for the **Search String** is selected from the **Volume to Search** list box by KwikINF or by you. KwikINF selects the **Volume to Search** by looking for the text string that has a matching entry in the KwikINF index file or, if there is no matching entry in the index file, in the **Default Volume** you have selected in the Configure KwikINF window. Or you can override KwikINF's selection of the **Volume to Search** by making your own selection from the list box.

You can also open and display the Contents of an online document by double-clicking on an entry in this list box.

KwikINF Keys Help

Use your KwikINF hot key (ALT+Q or the hot key you select from the Configure KwikINF window) to display the KwikINF window. You can also use your KwikINF hot key to initiate a search for a text string automatically, when you configure KwikINF to bypass the KwikINF window.

To re-configure KwikINF, when you have configured KwikINF to bypass the KwikINF window, press SHIFT + your KwikINF hot key.

To determine what your KwikINF hot key is, double-click on the KwikINF program object in the OS/2 Toolkit Information folder.

Link for Object and Library Files (LINK386)

LINK386 is used to combine object files and standard library files into a single file: an executable file, a dynamic-link library, or a device driver. The output file from LINK386 is not constrained to specific memory addresses. Thus, the operating system can load and execute this file at any convenient address.

LINK386 Input

LINK386 uses the following files as input:

- One or more object files that are linked with any optional library files to form the executable file. Object files usually have a .OBJ

extension.

LINK386 accepts object files compiled or assembled for 8088, 80286, 80386, 80486, or PENTIUM* microprocessors. Object files must be in the Object Module Format (OMF), which is based on the Intel* 8086 OMF, and Tool Interface Standards Portable Formats Specification.

- One or more library files. The library files contain object modules that are linked to the object files to form the executable file. Library files usually have a .LIB extension.

  Library files are used to resolve external references in your object files.

- A module definition file. The module definition file provides information to LINK386 about the executable file or dynamic link library file it is creating. The module definition file usually has a .DEF extension.

LINK386 Output

LINK386 can produce dynamic-link libraries (.DLL) and device drivers (.SYS), in addition to executable files (.EXE). For additional information, see Output Files.

LINK386 displays all of its output messages on the standard output device.

LINK386 Features

LINK386 creates the executable file and map file in the current directory unless you enter an explicit path.

LINK386 looks in several locations for object, library, and module-definition files. See Where LINK386 Looks for Files.

File names are not case sensitive; for example, abc.exe and ABC.EXE refer to the same file.

If you enter a file name without an extension, LINK386 adds a Default Filename Extension that depends on the type of file expected.

If you leave a field blank (but define the field with a comma), LINK386 uses a default for the field. If you end the LINK386 command with a semicolon (;), LINK386 uses Filename Defaults for all remaining fields.

If you do not give all file names or do not end the command line with a semicolon, LINK386 prompts you for the omitted files.

Starting LINK386

Some commands and applications call LINK386 for you, or you can run LINK386 by typing `LINK386` at the operating-system prompt. Supply input to LINK386 by any of three methods:

- Enter the input directly on the command line.
- Respond to prompts generated by LINK386.
- Put your input in a response file, and enter the file name on the command line.

You can press Ctrl+C at any time to interrupt LINK386 and return to the operating system.

To display LINK386 help, type `LINK386 /?` at the prompt. A copyright statement appears along with a list of valid LINK386 options.

Syntax

```
LINK386 [options] objfiles [,exefile, mapfile, libraries,
deffile]
```

OR

```
LINK386 @responsefile
```

SYNTAX DEFINITIONS

The LINK386 command line includes the following fields:

<options>
: Options modifying actions of LINK386. Options can appear anywhere on the command line except immediately after the commas used to separate fields. See Options and Using LINK386 Options.

<Object Files>
: Object files to be linked. Separate multiple file names by plus (+) or space characters. At least one name must be entered. Library files can also be entered. See Entering Library Files as Object Files.

<exefile>
: Output of file. LINK386 produces either an executable file, a dynamic-link library, or a device driver.

<mapfile>
: Map file created that lists modules in <exefile>. Use the /M option to include public symbols in this file. Enter NUL if you do not want a map file. See List Public Symbols (/M).

<libraries>
: Standard or import (not dynamic-link) libraries to be used in resolving external references. Separate multiple file names by plus (+) or space characters. Some libraries are searched by default. You can also specify a path to a directory -- LINK386 will search for libraries in a path specified on the <libraries> line before searching directories given by the LIB environment variable. See Linking with an Import Library, Default Libraries and Specifying Library Directories.

<deffile>
: Module definition file.

SYNTAX EXAMPLES

The following command links the object files FUN.OBJ, TEXT.OBJ, TABLE.OBJ, and CARE.OBJ. LINK386 searches for unresolved external references in the library file XLIB.LIB and in the default libraries. By default, the executable file is named FUN.EXE. LINK386 also produces a map file, FUNLIST.MAP.

```
LINK386 FUN+TEXT+TABLE+CARE, ,FUNLIST, XLIB.LIB;
```

The following command produces a map file named FUN.MAP because a comma appears as a placeholder for <mapfile>.

```
LINK386 FUN,,;
```

The next pair of command lines do not produce a map file because commas do not appear as placeholders for <mapfile>.

```
LINK386 FUN,;
LINK386 FUN;
```

The following command links the files MAIN.OBJ, GETDATA.OBJ, and PRINTIT.OBJ into an executable file named MAIN.EXE. A map file named MAIN.MAP is also produced.

```
LINK386 MAIN+GETDATA+PRINTIT, , MAIN;
```

The following command links GETDATA.OBJ and PRINTIT.OBJ into an OS/2 dynamic-link library. MODDEF.DEF must contain a LIBRARY statement to produce the dynamic-link library.

```
LINK386 GETDATA+PRINTIT,GETDATA.DLL, , MODDEF
```

Object Files

LINK386 accepts object files compiled or assembled for the 8088, 80286, 80386,80486 or PENTIUM microprocessor. LINK386 also accepts standard library files.

Output Files

LINK386 Can Produce Three Types of Output Files

- Executable (.EXE) files that run under OS/2 protected mode

  LINK386 produces an executable file whenever you specify a module-definition file containing a NAME Statement. The module definition file should not have a LIBRARY Statement, VIRTUAL DEVICE Statement, or PHYSICAL DEVICE Statement; otherwise, a dynamic-link library or device driver is produced, as described below.

- Dynamic-link library (.DLL) files

  A dynamic-link library is produced whenever you specify a module-definition file containing a LIBRARY statement.

- Device driver (.SYS) files

  A virtual or physical device driver is produced whenever you specify a module-definition file containing the VIRTUAL DEVICE or PHYSICAL DEVICE statements.

Prompts

LINK386 prompts you if any fields have not been entered on the Syntax or in a response file. For each prompt, simply enter the same input that you would enter on the command line and press Enter.

Object Modules [.OBJ]:
    <objfiles>

Run File [basename.EXE]:
    <exefile>

List File [NUL.MAP]:
    <mapfile>

Libraries [.LIB]:
    <libraries>

Definitions File [NUL.DEF]:
    <deffile>

Special Features

- To extend input to a new line, type a plus sign (+) as the last character on the current line. When the same prompt appears on a new line, you can continue. Do not, however, split a file name across lines.
- To select the default response to a prompt, press Enter. The next prompt appears.
- To select default responses to the current prompt and all remaining prompts, enter a semicolon (;). Note that at least one object file must be entered.
- You can specify options anywhere on any response line, except before a comma at the end of a line of characters. If you want to specify more than one option, either group them at the end of a response, or specify them at the end of several responses. Each option must begin with a slash (/).

Response Files

A response file is a text file used to provide input to LINK386. To use response file input for LINK386, type

```
LINK386 @responsefile
```

The @ symbol tells LINK386 that *responsefile* is the name of a response file. If the file is not in the working directory, you must specify the path.

The field <responsefile> specifies the name of a file containing the same input that would be entered on the command line or entered in response to LINK386 prompts. In this file, each response should appear on a separate line or be separated from other responses by a comma.

To operate LINK386 using a response file, you must first create a file that contains the responses you want LINK386 to process. You can give the file any name, and create it with any text editor.

Special Features:

- You can begin using a response file at any point on the LINK386 Syntax or at any LINK386 prompt. The response file should contain responses to all remaining fields or prompts.
- If the file does not contain responses for all the prompts, LINK386 displays the appropriate prompt and waits for you to supply a response. End the response file with a semicolon.
- You can use special characters in the response file the same way you would use them in responses entered at the keyboard. For example, you can extend input to a new line by using the plus sign (+) and choose default responses for all remaining prompts by using a semicolon (;).
- LINK386 displays prompts and the entries from the response file on the screen. If the entry in the response file is not acceptable, LINK386 pauses and waits for you to enter an acceptable response. The "Run in Batch Mode (/BAT)" disables the prompt.
- Options can appear anywhere in the response file.

Response File Example

```
FUN TEXT TABLE CARE
/DEBUG /MAP
FUNLIST
GRAF.LIB
```

If the text file above were named FUN.LNK, the following command would use this file as a response file:

```
LINK386 @FUN.LNK
```

This would cause LINK386 to do the following:

- Link the four object modules FUN, TEXT, TABLE, and CARE into an executable file named FUN.EXE
- Generate the map file FUNLIST.MAP
- Generate Debugging information
- Include public symbols and addresses in the map file
- Link any needed routines from the library file GRAF.LIB

The response file in the following example instructs LINK386 to generate an executable file, called FUN.EXE, from four object modules, FUN, SUN, RUN, and GAMES.

If you specify the file name, FUNLIST, LINK386 will generate a map file named FUNLIST.MAP. Adding the /MAP option will cause LINK386 to include the public symbols of the application in the map file.

```
fun+sun+run+game /map
```

```
fun.exe
funlist
;
```

Default Libraries

Most compilers embed the names of needed libraries (called default libraries) in object files. LINK386 searches these libraries. Because of this, you need to explicitly enter library names only in the following cases:

- You want to use additional libraries.
- You are using a library not in the current directory and not in a directory specified by the LIB environment variable. See Where LINK386 Looks for Files.
- You want to use a library other than the one specified in the object file.

Explicitly entered libraries are always searched before default libraries. If an external reference is resolved by more than one library, the order of libraries on the command line determines which library is used.

To ignore default libraries use the Ignore Default Libraries (/NOD). But be careful - most compilers expect their object files to be linked with default libraries.

Entering Library Files as Object Files

You can enter library files in the <objfiles> field. Be sure to include the .LIB file name extension; otherwise, LINK386 assumes a .OBJ extension.

With libraries entered in the <objfiles> field, LINK386 adds every module in the library to your output file. With libraries entered in the <libraries> field, LINK386 adds only those required to resolve external references.

The effect of entering a library this way is the same as if you had entered all of the library's module names into the <objfiles>field.

Specifying Library Directories

LINK386 searches additional locations for libraries using the drive name or path specification in the <libraries> field on the command line.

You can specify up to 32 additional paths. If you give more than 32 paths, LINK386 ignores the additional paths without displaying an error message.

Where LINK386 Looks for Files

When searching for an object, library, or module definition file, LINK386 looks in the following locations in this order:

1. The directory specified for the file if a path specification is included. Default Libraries do not include path specifications.
2. The current directory.
3. Any directories entered on the command line.
4. Any directories given by the LIB environment variable.

If LINK386 cannot locate a file, it prompts you to enter the location. The "Run in Batch Mode (/BAT)" disables these prompts.

Library Search Example

```
LINK386
Object Modules [.OBJ]: FUN TEXT TABLE CARE
Run File [FUN.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]: C:\TESTLIB\ NEWLIBV3
Definitions File [NUL.DEF]:
```

This example links four object modules to create an executable file named FUN.EXE. LINK386 searches NEWLIBV3.LIB before searching the default libraries to resolve references.

To locate NEWLIBV3.LIB and the default libraries, LINK386 searches the following locations in this order:

1. The current directory
2. The C:\TESTLIB\ directory
3. The locations given by the LIB environment variable

Filename Defaults

If you do not enter a file name, LINK386 assumes a default:

<options>
: No options

<objfiles>
: None (This field is required.)

<exefile>
: The base name of the first file in <objfiles> with the .EXE extension added

<mapfile>
: The base name in <exefile> with the .MAP extension added

<libraries>
: No libraries

<deffile>
: No module definition file

Default Filename Extension

If you do not enter an extension, LINK386 uses a default extension, depending on the type of file.

Object	.OBJ
Executable	.EXE
Map	.MAP
Standard Library	.LIB

Dynamic-Link Library	.DLL
Module Definition	.DEF

Overriding Default Extension

Any time you explicitly enter an extension, it overrides the default extension. To specify a file name without an extension, just enter a period (.) after the file name.

Options

The following is a summary of LINK386 options:

/?	Display Help
/A[LIGNMENT]	Align
/BAS[E]	Base
/BAT[CH]	Run in Batch Mode
/C[ODEVIEW]	Prepare for Debugging
/DE[BUG]	Prepare for Debugging
/DO[SSEG]	Order Segments
/E[XEPACK]	Exepack (You can add :1 or :2.)
/EXEPACK	Exepack
/EXEPACK:1	Exepack
/E:2	Exepack
/E:1	Exepack
/EXE:2	Exepack
/EXEP	Exepack
/F[ARCALLTRANSLATION]	Optimize Far Calls
/H[ELP]	Display Help
/I[NFORMATION]	Display Process Information
/L[INENUMBERS]	Include Line Numbers
/M[AP]	List Public Symbols
/NOD[EFAULTLIBRARYSEARCH]	Ignore Default Libraries
/NOE[XTDICTIONARY]	Ignore Extended Dictionary
/NOF[ARCALLTRANSLATION]	Disable Far Optimization
/NOI[GNORECASE]	Preserve Case Sensitivity
/NOL[OGO]	Disable Sign-On Banner
/NON[ULLSDOSSEG]	Order Segments without NULLs
/NOO[UTPUTONERROR]	Does not produce <exefile> if an error occurs during linking.
/NOS[ECTORALIGNCODE]	Disable Automatic Sector Alignment code.
/NOP[ACKCODE]	Disable Code-Segment Packing
/PACKC[PACKCODE]	Combine Contiguous Code
/PACKD[ATA]	Combine Contiguous Data
/PAU[SE]	Pause during Linking
/PM[TYPE]	Name Application Type
/RU[NFROMVDM]	Allow Execution From DOS Command Line
/SE[GMENTS]	Set Max Number of Segments
/ST[ACK]	Control Stack Size
/W[ARNFIXUP]	Warn Fixup

Options Not Supported Under LINK386

/O[VERLAYINTERRUPT]
/CP[ARMAXALLOC]
/PADC[ODE]
/DS[ALLOCATE]
/PADD[ATA]
/Q[UICKLIB]
/HI[GH]
/T[INY]
/INC[REMENTAL]
/NOG[ROUPALIGN]

Specifying LINK386 Options

You can specify options anywhere on the response line, except before a comma at the end of a line of characters. If you want to specify more than one option, either group them at the end of a response, or specify them at the end of several responses. Each option must begin

with a forward slash (/).

Using LINK386 Options

1. Options always begin with the slash character (/).
2. Options are not case sensitive. For example, /de and /DE are equivalent.
3. You can specify options in either the short or long form. The short form is the shortest sequence of characters that uniquely identifies the option. The individual description of each option lists both forms with the optional part enclosed in brackets. For example, /BAT[CH] indicates that either /BAT or /BATCH can be used.
4. Some linker options take numeric arguments. You can enter numbers in decimal, octal, or hexadecimal radix using standard C-language syntax.
5. You can also specify options in the LINK386 environment variable.
6. Although use of the minimum one-letter abbreviations is allowed, if a future release has an additional option starting with the same letter, the one-letter option will no longer be usable.

Recommendations

It is recommended that the ALIGN:2, BASE, FARCALLTRANSLATION, RUNFROMVDM, and EXEPACK:2 options be used when linking all executables. This will compress them in size and improve their performance. Executables linked with EXEPACK:2 can be run only on OS/2 versions 3.0 and later.

If BASE is used with .EXE files, the /BASE:0x10000 option must be used. Any other value will produce a warning.

Entering Numeric Arguments

Some LINK386 options and module statements take numeric arguments. LINK386 uses C-language syntax allowing you to specify numbers in any of the following forms:

- Any number not prefixed with 0 or 0x is a decimal number. For example, 1234 is a decimal number.
- Any number prefixed with 0 (but not 0x) is an octal number. For example, 01234 is an octal number.
- Any number prefixed with 0x is a hexadecimal number. For example, 0x1234 is a hexadecimal number.

Environment Variable

You can use the LINK386 environment variable to cause certain options to be used each time you link. LINK386 checks the environment variable for options if the variable exists.

LINK386 expects to find options listed in the variable exactly as you would type them on the command line. It does not accept other kinds of arguments; file names in the environment variable cause the following error message:

```
unrecognized option
```

Each time you link, you can specify other options in addition to the ones specified in the LINK386 environment variable. If you type an option both on the Syntax and in the environment variable, the effect is the same as if the option were given once.

Note: A command line option overrides the effect of any environment-variable option that it conflicts with. For example, the command line option /SE:512 cancels the effect of the environment-variable option /SE:256.

The only way to prevent an option in the environment variable from being used is to reset the environment variable itself.

Environment Variable Example

```
<SET LINK386=/NOI /SE:256 /DEBUG
<LINK386 TEST;
<LINK386 /NOD /DEBUG PROG;
```

In the example above, the file TEST.OBJ is linked with the options /NOI, /SE:256, and /DEBUG. The file PROG.OBJ is then linked with the option /NOD - in addition to /NOI, /SE:256, and /DEBUG.

Alignment (/A)

Syntax: `/A[LIGNMENT]:n`

This option directs LINK386 to set the alignment factor in the executable file to the number given, which must be a power of 2, from 2 to 32768. The default alignment is 512 bytes. Trailing zeroes are truncated to reduce the amount of data stored in a file.

Each page starts at a location that is a multiple of *n* bytes from the beginning of the file. For example, `/A:16` would start pages at multiples of 16 bytes.

Link386 produces pages that are a maximum of 4096 bytes in length. Alignment factors greater than 4096 will waste disk space.

BASE (/BASE)

Syntax: `BASE:n`

Where n is a value rounded up to the nearest multiple of 64K Indicates that each object of the module has a preferred load address starting with object 1 at this address, object 2 at the next available multiple of 64K, and so on. Internal relocation records are then applied using this addressing scheme.

If the module's objects can be loaded beginning at this preferred address, then no load-time internal relocation records need be applied.

If the module's objects cannot be loaded beginning at this preferred address, then the internal relocation records that have been retained in the file data will be applied.

.EXE files may specify a base address, but it must be 64K. If it isn't, a warning will be issued and a base address of 64K will be used anyway. This option provides the same support as the BASE module definition file statement.

Run in Batch Mode (/BAT)

Syntax: `/BAT[CH]`

By default, LINK386 prompts you for a new path name whenever it cannot find an object file or library it was directed to use.

This option disables such prompting. Instead, LINK386 generates an error or warning message, as appropriate, and leaves the external reference unresolved. The /BAT option also disables the display of the sign-on banner and the display of input from response files.

This option is primarily used when LINK386 is called from a batch file or Description Files.

Note: This option does not affect prompts for Command Line Input.

Prepare for Debugging (/C)

Syntax: `/C[ODEVIEW]`

This option works exactly like the Prepare for Debugging (/DE) option.

The /C option is used to prepare for debugging with any debugger. With this option, LINK386 imbeds symbolic data and line number information in the executable output file.

You can run this executable file outside Debug; the debugging information in the file is ignored. However, to reduce executable file size, use this option only for debugging. Then you can link a separate version without the /C option after the program is debugged.

Prepare for Debugging (/DE)

Syntax: `/DE[BUG]`

The /DE option is used to prepare for debugging with any debugger. With this option, LINK386 embeds symbolic data and line number information in the executable output file.

You can run this executable file outside Debug; the debugging information in the file is ignored. However, to reduce executable file size, use this option only for debugging. Then you can link a separate version without the /DE option after the program is debugged.

Order Segments (/DO)

Syntax: `/DO[SSEG]`

This option is automatically enabled by a special object module record in many language libraries. If you are linking to one of these libraries, you need not specify this option.

The /DO option is also enabled by assembly modules that use the Macro Assembler directive .DOSSEG.

This option forces segments to be ordered as follows (first to last):

1. All code segments
2. Far data segments
3. Near data (DGROUP) segments, in the following order:
    a) Any segments of class BEGDATA (this class name is reserved)
    b) Any segments not of class BEGDATA, BSS, or STACK
    c) Segments of class BSS
    d) Segments of class STACK

In addition, the /DO option causes LINK386 to do the following:

- Initialize two special variables:

  ```
  _edata = DGROUP : BSS
  _end = DGROUP : STACK
  ```

  The variables _edata and _end have special meanings for certain compilers; avoid using these names in your programs. Assembly-language programs can refer to these variables, but should not change them.

- Insert 16 null bytes at the beginning of the _TEXT segment (if this segment is defined).

Exepack (/E)

Syntax: `/E[XEPACK] or /E[EXEPACK]:1 or /E[XEPACK]:2.`

EXEPACK causes pages of code and data in the file to be compressed. The OS/2 Application Loader will automatically decompress these pages when the program is run.

/EXEPACK:1 will use a compression algorithm that is compatible with OS/2 2.0, 2.1, and 2.11, as well as OS/2 3.0 and later.

/EXEPACK:2 will use a compression algorithm that is compatible with OS/2 3.0 and later.

/EXEPACK:2 will produce smaller executables that typically load faster.

Optimize Far Calls (/F)

Syntax: `/F[ARCALLTRANSLATION]`

This option causes LINK386 to optimize far-call instructions made from one segment to a target address in the same segment. LINK386 replaces calling sequences such as `CALL FAR function` with the following:

```
PUSH     CS
CALL     NEAR function
NOP
```

The new calling sequence is significantly faster when running in protected mode. Also, a load-time relocation is eliminated, which decreases program file size and speeds program loading.

In general, the greatest benefit occurs if you use the "Combine Contiguous Code(/PACKC)" in addition to the /F option.

The /F option has no effect on programs that make only near calls.

Note: There is a small risk involved with using the /F option. LINK386 may mistakenly interpret a byte of immediate data in a code segment as a far call if it has to have the far-call opcode (0x9A).

Display Help (/H or /?)

Syntax: `/H[ELP]` OR `/?`

These options display a list of valid LINK386 options.

Display Process Information (/I)

Syntax: `/I[NFORMATION]`

This option causes LINK386 to display information about the linking process, including the phase of linking and the names of the object files being linked. Use this option to determine the locations of the object files being linked and the order in which they are linked.

The output from this option is sent to standard output.

Include Line Numbers (/L)

Syntax: `/L[INENUMBERS]`

This option includes source file line numbers and associated addresses in the map file. In addition, you must give LINK386 an object file (or files) with line number information. You can use the /Zd option with most compilers to include line numbers in the object file. If you give LINK386 an object file without line number information, the /L option has no effect. The option for CSET/++ is /ti.

The /L option forces LINK386 to create a map file even if you did not explicitly tell LINK386 to create a map file. By default, the file is given the same base name as the executable file, plus the extension .MAP. You can override the default name by explicitly specifying a map file name.

List Public Symbols (/M)

Syntax: `/M[AP][:full]`

This option lists in the map file all public (global) symbols defined in the object files. With this option, the map file contains a list of all the symbols sorted by name, and a list of all the symbols sorted by address. If you don't use this option, the map file contains only a list of segments.

With this option, LINK386 creates a map file by default. If you explicitly enter a map file name of NUL, then no map file is created, and this option has no effect.

The Map option can be specified as `/M:full` to produce a comprehensive map showing the composition of each segment.

Ignore Default Libraries (/NOD)

Syntax: `/NOD[EFAULTLIBRARYSEARCH][:filename]`

This option tells LINK386 to ignore Default Libraries when resolving external references. If you specify an object file in <filename>, LINK386 ignores only the default libraries in <filename>.

In general, high-level-language programs do not work correctly without standard libraries. Thus, if you use the /NOD option, you should explicitly specify the name of a standard library in the <libraries> field of the command line.

Ignore Extended Dictionary (/NOE)

Syntax: `/NOE[XTDICTIONARY]`

This option prevents LINK386 from searching the *extended dictionary* , an internal list of symbol locations included with libraries generated

with the old LIB utility's /NOE option.

Normally, LINK386 uses the extended dictionary to speed up library searches; thus, using /NOE slows LINK386. This option should be used when a library symbol is redefined. You need to use this option when LINK386 issues error L2044.

Disable Far Optimization (/NOF)

Syntax: /NOF[ARCALLTRANSLATION]

Far-call optimization is off by default. If the LINK386 environment variable or another command (such as ICC) has turned it on, you can use /NOF to turn it off again.

Preserve Case Sensitivity (/NOI)

Syntax: /NOI[GNORECASE]

This option turns case sensitivity on; that is, LINK386 treats ABC, abc, and Abc as unique names. By default, case sensitivity is off.

This option can be used when you link programs written in case-sensitive languages such as C.

Disable Sign-On Banner (/NOL)

Syntax: /NOL[OGO]

This option disables the sign-on banner displayed when LINK386 starts.

Order Segments without NULLs (/NON)

Syntax: /NON[ULLSDOSSEG]

This option arranges segments in a special order. The /NON option is equivalent to the Order Segments (/DO) except that /NON does NOT insert 16 null bytes at the beginning of the _TEXT segment (if this segment is defined).

The /NON option overrides the /DO option when both are used. Therefore, you can use /NON to override the /DO comment record commonly found in standard libraries.

No Output on Error (/NOO)

Syntax: /NOO[UTPUTONERROR]

This option will keep Link386 from creating the executable if an error is encountered.

Disable Code-Segment Combining (/NOP)

Syntax: `/NOP[ACKCODE]`

This option turns code segment combining off. By default, code segment combining is on.

Disable Sector Alignment of Code (/NOS)

Syntax: `/NOS[ECTORALIGNCODE]`

Link386 will align passes of code on sector (512 bytes) boundaries. This helps reduce the time to load the passes. The /NOSECTORALIGNCODE option is provided to turn off this feature. Pages of code would then be aligned based on the /ALIGN value.

Combine Contiguous Code (/PACKC)

Syntax: `/PACKC[ODE]:number`

This option groups neighboring code segments. Neighboring code segments are assigned the same segment address, and offsets to each routine are adjusted upward as required. This option is on by default and is used only when you wish to override an environment variable that has turned code combining off. See Environment Variable and Disable Code-Segment Combining (/NOP)

The <number> field specifies the maximum size of a code segment grouped by /PACKC. If you do not use the /PACKC option or if you omit <number>, maximum size defaults to 65530. LINK386 stops adding segments to a group as soon as it cannot add another segment without exceeding <number>. At this point, LINK386 forms a new segment. See Entering Numeric Arguments.

Code combining generally produces slightly faster and more compact code. Use the Optimize Far Calls (/F) option to provide the maximum opportunity for combining.

Combining Contiguous Data (/PACKD)

Syntax: `/PACKD[ATA] [:number]`

This option groups neighboring data segments. It functions like the Combine Contiguous Code (/PACKC) option, except that it combines data segments.

The <number> field specifies the maximum size of a data segment grouped by /PACKD; if you omit <number>, the maximum size defaults to 65,536. LINK386 stops adding segments to a group as soon as it cannot add another segment without exceeding <number>. At this point, LINK386 forms a new group. See Entering Numeric Arguments.

Pause during Linking (/PAU)

Syntax: `/PAU[SE]`

This option makes LINK386 pause before writing the output file to disk. The pause allows you to swap disks.

With this option, LINK386 displays the following message before it creates the output file:

```
About to generate .EXE file
Change diskette in drive letter and press Enter
```

LINK386 writes the output file when you press Enter.

Be sure not to remove a disk containing the map file. If the disk you need to swap contains either of these files, press CTRL+C to terminate the LINK386 session, rearrange your files, and link again.

Name Application Type (/PM)

Syntax: `/PM[TYPE]:type`

This option specifies the type of application being generated. Using the /PM option is equivalent to including a NAME statement in the module definition file.

A keyword in <type> is equivalent to a keyword in a NAME statement as shown in the following list:

PM	WINDOWAPI
VIO	WINDOWCOMPAT
NOVIO	NOTWINDOWCOMPAT

Execute from DOS Command Line (/RU)

Syntax: `/RU[NFROMVDM]:`

This option allows the program to be executed from a DOS command line, if possible.

This option causes LINK386 to insert an alternate DOS stub into the program. The DOS stub is executed if a protect mode program is executed from a DOS command line. The default DOS stub simply prints an error message and returns to the DOS command line. The alternate DOS stub will attempt to start the program in protect mode.

Set Maximum Number of Segments (/SE)

Syntax: `/SE[GMENTS]:number`

This option sets the number of logical segments a program can have. You can set <number> to any value in the range 1 to 3,072. See Entering Numeric Arguments.

For each logical segment, LINK386 must allocate space to keep track of segment information. By using a relatively low segment limit as a default (128), LINK386 is able to link faster and allocate less storage space.

When you set the segment limit higher than 128, LINK386 allocates more space for segment information. This option allows you to raise the segment limit for programs with a large number of segments.

For programs with fewer than 128 segments, you can keep the storage requirements of LINK386 at the lowest level possible by setting the segment <number> field to reflect the actual number of segments in the program. If the number of segments allocated is too high for the amount of memory LINK386 has available to it, you see the error message `segment limit too high`.

To specify a number of segments that fits in the amount of memory available, set the segment lower and relink.

Control Stack Size (/ST)

Syntax: `/ST[ACK] :number`

This option controls the stack size (in bytes) of your program. You can specify any positive value for <number>. See Entering Numeric Arguments.

If your program generates a stack-overflow message, you can increase the size of the stack. In contrast, if your program uses the stack very little, you may save some space by decreasing the stack size.

Warning of Fix-ups (/W)

Syntax: `/W[ARNFIXUP]`

This option directs LINK386 to issue a warning for each segment-relative fix-up of location-type offset when the segment is contained within a group, but not at the beginning. LINK386 includes the displacement of the segment from the group in determining the final value of the fix-up.

/INF and /M Output Example

The following is a sample of LINK386 output when Display Process Information (/I) and List Public Symbols (/M) options are specified:

```
**** PASS ONE ****
TEST.OBJ(test.for)
**** LIBRARY SEARCH ****
LLIBFOR7.LIB(wr)
LLIBFOR7.LIB(fmtout)
LLIBFOR7.LIB(ldout)

**** ASSIGN ADDRESSES ****
1 segment "TEST_TEXT" length 122H bytes
2 segment "_DATA" length 912H bytes
3 segment "CONST" length 12H bytes

**** PASS TWO ****
TEST.OBJ(test.for)
LLIBFOR7.LIB(wr)
LLIBFOR7.LIB(fmtout)
LLIBFOR7.LIB(ldout)

**** WRITING EXECUTABLE ****
```

OS/2 Considerations

In most respects, linking a program for OS/2 is similar to linking a program for DOS. The principal difference is that most programs created for DOS run as stand-alone applications, whereas programs for OS/2 generally call one or more *dynamic-link libraries.* See What Is a Dynamic-Link Library? and Advantages of Dynamic Linking.

Import and Export Definitions

Each dynamic-link library (.DLL file) defines *export definitions* that tell OS/2 what functions the library has. Functions not exported can only be called from within the library. Each export definition specifies a function name.

Conversely, each executable program (.EXE file) defines *import definitions* that tell OS/2 which dynamic-link functions the program needs and where they can be found. Otherwise, OS/2 would not know which dynamic-link libraries to load when the program is run. Each import definition specifies a function name and the .DLL file where the function resides.

Methods of OS/2 Linking

There are two methods of OS/2 Linking:

- Linking without an Import Library
- Linking with an Import Library

Linking with an import library requires more steps but has certain advantages.

What Is a Dynamic-Link Library?

A dynamic-link library contains executable code for common functions, just as an ordinary library does. Yet code for functions in dynamic-link libraries is not copied into the executable (.EXE) file. Instead, the library itself is loaded into memory at run time, along with the .EXE file.

Advantages of Dynamic Linking

Dynamic-link libraries serve much the same purpose that standard libraries do, but they also have the following advantages:

- **Applications link more quickly**. With dynamic linking, the executable code for a dynamic-link function is not copied into the .EXE file of the application. Instead, only an import definition is copied.
- **Applications require less disk space.** With dynamic linking, several different program applications can access the same dynamic-link function stored in one place. Without dynamic linking, the code for the function would be repeated in every .EXE file.
- **Libraries and applications are independent.** Dynamic-link libraries can be updated any number of times without relinking the applications that use them. If you are a user of third-party libraries, this is particularly convenient. You receive the updated .DLL file from the third-party developers, and you only need to copy the new library onto your disk. At run time, your applications automatically call the updated library functions.
- **Code and data segments can be shared.** Code and data segments loaded from a dynamic-link library can be shared. Without dynamic linking, such sharing is not possible because each file has its own copy of all the code and data it uses. By sharing segments with dynamic linking, you can use memory more efficiently.

Linking without an Import Library

The figure below illustrates a simple case in which you create an application that uses a single dynamic-link library (.DLL) file.

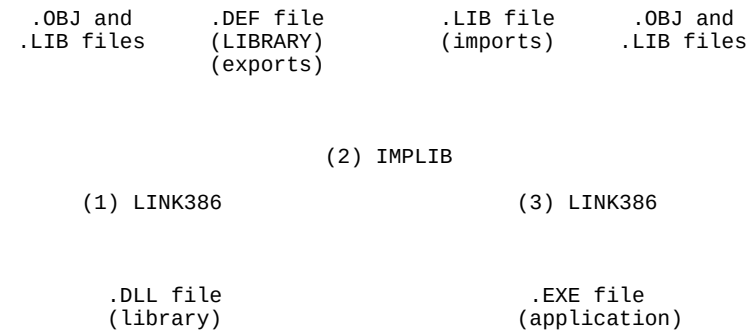
As depicted above, linking occurs in two steps:

1. Object files (and standard libraries if any) are linked with a module definition (.DEF) file to create a .DLL file. A .DEF file is used that defines all functions exported by the .DLL file.

2. Object files (and standard libraries, if any) are linked with a .DEF file to create an application (.EXE) file. A different .DEF file is used for this step; it defines all dynamic-link functions imported (used) by the application.

Linking with an Import Library

The figure below illustrates a simple case in which you create an application that uses a single dynamic-link library (.DLL) file.



As depicted above, linking occurs in three steps:

1. Object files (and dynamic-link libraries) are linked with a module definition (.DEF) file to create a .DLL file. A .DEF file is used that defines all functions exported by the .DLL file.

2. The IMPLIB program is used to generate an *import library* (.LIB) file. IMPLIB takes as input the same module definition file used in the first step. For each export definition in the .DEF file, IMPLIB generates a corresponding import definition. (IMPLIB can also use the .DLL file generated in step 1 if you use the _export keyword in C declarations to export functions.)

3. The .LIB file generated by IMPLIB is used as input to LINK386, which creates an application (.EXE) file. This .LIB file provides LINK386 with information about imported dynamic-link functions.

Module Definition Files Basics

What Is a Module Definition File?

A module definition file describes the names, attributes, exports, imports, and other characteristics of an application or library. You must use module definition files to create most applications for OS/2. You must use module definition files to create all OS/2 dynamic-link libraries and device drivers.

Module Statements

A module definition file contains one or more *Module Statements* These statements:

- Define various attributes of the executable file
- Define attributes of code and data segments
- Identify functions that are imported or exported

Module Definition File Example

The following module definition file gives module definitions for a dynamic-link library. It includes one source level comment and five statements.

```
; Sample module-definition file

LIBRARY

DESCRIPTION
 'Sample .DEF file for a dynamic-link library'

CODE      PRELOAD

STACKSIZE 1024

EXPORTS
 Init    @1
 Begin   @2
 Finish  @3
 Load    @4
 Print   @5
```

Module Statement Rules

- If you use a NAME, LIBRARY, VIRTUAL DEVICE, or PHYSICAL DEVICE statement, it must precede all other statements in the module definition file.
- You can include source level comments in the module definition file by beginning a line with a semicolon (;). Such lines are ignored.
- All module definition keywords (such as NAME, LIBRARY, and OLD) must be entered in uppercase letters.

Module Statements

LINK386 has the following module definition file statements:

BASE	Base
CODE	Gives default attributes for code segments
DATA	Gives default attributes for data segments
DESCRIPTION	Describes the module
EXETYPE	Identifies operating system
EXPORTS	Defines exported functions
IMPORTS	Defines imported functions
HEAPSIZE	Specifies local heap size
LIBRARY	Names dynamic-link library
NAME	Names application
OLD	Preserves import information
PHYSICAL DEVICE	Names physical device driver
PROTMODE	Specifies DOS protected mode
SEGMENTS	Gives attributes for specific segments
STACKSIZE	Specifies local stack size

STUB	Adds DOS executable file to module
VIRTUAL DEVICE	Names virtual device driver

BASE Statement

Syntax: BASE=n

Where n is a value rounded up to the nearest multiple of 64K Indicates that each object of the module has a preferred load address starting with object 1 at this address, object 2 at the next available multiple of 64K, and so on. Internal relocation records are then applied using this addressing scheme.

If the module's objects can be loaded beginning at this preferred address, then no load-time internal relocation records need be applied.

If the module's objects cannot be loaded beginning at this preferred address, then the internal relocation records that have been retained in the file data will be applied.

EXE files may specify a base address, but it must be 64K. If it isn't, a warning will be issued and a base address of 64K will be used anyway.

CODE Statement

Syntax: `CODE [attribute...]`

This statement defines the default attributes for code segments within the application or library. One or more attributes can appear following the CODE statement:

PRELOAD or LOADONCALL
: Sets when code segment is loaded

  Note: OS/2 2.x ignores the preload attribute.

EXECUTEONLY or EXECUTEREAD
: Sets read/execute status

IOPL or NOIOPL
: Sets I/O privilege

CONFORMING or NONCONFORMING
: Determines segment conformance

Attribute Rules:

- Only one attribute from each pair appears. If you specify neither attribute from a pair, LINK386 supplies the default, listed second in each pair above.
- Attributes can appear in any order.

Example

The following example sets defaults for module code segments so they have I/O hardware privilege and are not loaded until accessed.

```
CODE LOADONCALL IOPL
```

Load Code Attributes

These attributes determine when a code segment is loaded:

PRELOAD	The segment is loaded automatically when the program starts. **Note:** OS/2 2.x ignores the preload attribute.
LOADONCALL	The segment is not loaded until accessed (default).

Read/Execute Code Attributes

These attributes determine whether a code segment can be read as well as executed:

EXECUTEONLY	The segment can only be executed.
EXECUTEREAD	The segment can be both executed and read (default).

I/O Privilege Code Attributes

I/O privilege code attributes determine whether a segment has I/O privilege (that is, whether it can access the hardware directly):

IOPL
: The code segment has I/O privilege.

NOIOPL
: The code segment does not have I/O privilege (default).

Conforming Code Attributes

These attributes specify whether a code segment is a conforming segment:

CONFORMING	The segment is conforming.
NONCONFORMING	The segment is nonconforming (default).

The concept of a conforming segment has to do with privilege level (the range of instructions that the process can execute) and is relevant only when you are writing device drivers and system level code. A conforming segment can be called from either Ring 2 or Ring 3, and the segment executes at the privilege level of the caller.

DATA Statement

Syntax: `DATA [attribute...]`

This statement defines the default attributes for data segments within the application or library. One or more attributes can appear following the DATA statement:

PRELOAD or LOADONCALL
: Sets when data segment is loaded

  Note: OS/2 2.x ignores the preload attribute.

READONLY or READWRITE
Sets read/write access

NONE, SINGLE, or MULTIPLE
Sets sharing attributes

IOPL or NOIOPL
Sets I/O privilege

SHARED or NONSHARED
Determines whether segment is shared

Attribute Rules

- Only one attribute out of each group appears; if you specify none of the attributes in a group, the last attribute listed above for that group is generally the default. (The defaults for NONE/SINGLE/MULTIPLE and SHARED/NONSHARED vary depending on whether you are describing a dynamic-link library or application.)
- Attributes can appear in any order.

Example

The following example defines the application data segment so that it is loaded only when it is accessed and cannot be shared by more than one copy of the program. By default, the data segment can be read and written, the automatic data segment is copied for each instance of the module, and the data segment has no I/O privilege.

```
DATA LOADONCALL NONSHARED
```

Load Data Attributes

These attributes determine when a data segment is loaded:

PRELOAD	The segment is loaded automatically when the program starts. **Note:** OS/2 2.x ignores the preload attribute.
LOADONCALL	The segment is not loaded until accessed (default).

Read/Write Data Attributes

These attributes determine the access rights to a data segment:

READONLY	The segment can only be read.
READWRITE	The segment can be both read and written to (default).

Sharing Data Attributes

These attributes determine how the automatic data segment can be shared:

NONE	No automatic data segment is created.
SINGLE	A single automatic data segment is shared by all instances of the module. In this case, the module is said to have

solo data. This keyword is the default for dynamic-link libraries.

MULTIPLE — The automatic data segment is copied for each instance of the module. In this case, the module is said to have *instance* data. This keyword is the default for applications.

These attributes refer to initialized global data. The automatic data segment is the physical segment represented by the group name DGROUP. This segment group makes up the physical segment that contains the local stack and heap of the application.

Shareable Data Attributes

These attributes determine whether all instances of the program can share a READWRITE data segment:

SHARED — One copy of the data segment is loaded and shared among all processes accessing the module (default for dynamic-link libraries).

NONSHARED — The segment cannot be shared, and must be loaded separately for each process (default for applications).

These attributes refer to non-initialized global data. Under OS/2, this field is ignored if READONLY is specified, since READONLY data segments are always shared.

LINK386 -I/O Privilege Data Attributes

These attributes determine whether data segments have I/O privilege (that is, whether they can access the hardware directly):

IOPL — The data segments have I/O privilege.

NOIOPL — The data segments do not have I/O privilege (default).

SEGMENTS Statement

Syntax:

```
SEGMENTS
         segmentdefinitions
```

This statement defines the attributes of one or more segments in the application or library on a segment-by-segment basis. The attributes specified by this statement override defaults set in CODE and DATA statements.

The SEGMENTS keyword marks the beginning of the segment definitions. This keyword can be followed by one or more segment definitions, each on a separate line (limited by the number set by the LINK386 /SE option, or 128 if the option is not used). See Set Maximum Number of Segments (/SE).

Segment-Definition Syntax

```
['] segmentname ['] [CLASS 'classname'][attribute...]
```

Each segment definition begins with <segmentname>, optionally enclosed in single quotation marks ('). The quotation marks are required if <segmentname> conflicts with a module definition keyword, such as CODE or DATA.

The CLASS keyword specifies the class of the segment. Single quotation marks (') are required for <classname>. If you do not use the CLASS argument, LINK386 assumes that the class is CODE.

One or more attributes can follow. The default attribute is listed last.

PRELOAD or LOADONCALL
: Determines when segment is loaded

  Note: OS/2 s.x ignores the preload attribute.

READONLY or READWRITE
: Sets read/write access

EXECUTEONLY or EXECUTEREAD
: Sets read/execute status

IOPL or NOIOPL
: Sets I/O privilege

CONFORMING or NONCONFORMING
: Determines conformance

MIXED1632
: Specify Mixed 16 and 32-Bit Segments

ALIAS
: Specify that segment is aliased

SHARED or NONSHARED
: Specify that segment is shared

Attribute Rules

- Only one attribute from each pair appears. If you specify neither attribute from a pair, LINK386 supplies the default (listed second in each pair above).
- Attributes can appear in any order.

Example

```
SEGMENTS
  cseg1 CLASS 'mycode' IOPL
  cseg2 EXECUTEONLY PRELOAD CONFORMING
  dseg CLASS 'data' LOADONCALL READONLY
```

This example specifies segments named cseg1, cseg2, and dseg. The first segment is assigned class *mycode* and the second is assigned CODE by default. Each segment is given different attributes.

Load Segments Attributes

These attributes determine when a segment is loaded:

PRELOAD	The segment is loaded automatically when the program starts. **Note:** OS/2 2.x ignores the preload attribute.
LOADONCALL	The segment is not loaded until accessed (default).

Read/Write Segments Attributes

These attributes determine the access rights to a data segment:

READONLY	The segment can only be read.

READWRITE	The segment can be both read and written to (default).

Read/Execute Segments Attributes

These attributes determine whether a code segment can be read as well as executed:

EXECUTEONLY	The segment can only be executed.
EXECUTEREAD	The segment can be both executed and read (default).

I/O Privilege Segments Attributes

These attributes determine whether a segment has I/O privilege (that is, whether it can access the hardware directly):

IOPL	The segment has I/O privilege.
NOIOPL	The segment does not have I/O privilege (default).

Conforming Segments Attributes

These attributes specify whether a code segment is a conforming segment:

CONFORMING	The segment is conforming.
NONCONFORMING	The segment is nonconforming (default).

The concept of a conforming segment has to do with privilege level (the range of instructions that the process can execute). Conforming attributes are relevant only when you are writing device driver and system-level code. A conforming segment can be called from either Ring 2 or Ring 3, and the segment executes at the privilege level of the caller.

Specify Mixed 16 and 32-Bit Segments

Sometimes it is necessary to mix 16-bit code with 32-bit code. When you must create groups that allow such mixing, LINK386 requires that you declare the segments in that group as MIXED1632.

Specify that Segment is Aliased

Segments flagged with the ALIAS keyword can be addressed using the 16-bit segmented method (_far16), or the 32-bit linear method. The loader must prepare an additional segment selector for each segment designated with the ALIAS keyword. This new segment selector allows for 16-bit addressing.

Example:

```
SEGMENTS _CODE  ALIAS
```

The statement above specifies that the segment *_CODE* can be called using 16-bit far calls and 32-bit near calls.

Specify that Segment is Shared

These attributes determine if the segment can be shared by other processes.

SHARED	One copy of the data segment is loaded and shared among all processes accessing the module (default for dynamic-link libraries).
NONSHARED	The segment cannot be shared, and must be loaded separately for each process (default for applications).

DESCRIPTION Statement

Syntax: `DESCRIPTION 'text'`

This statement inserts the specified text into the application or library. The DESCRIPTION statement is useful for embedding source control or copyright information into an application or library.

The <text> field is a one line string enclosed in single quotation marks.

Example

The following example inserts the text *Template Program* into the application or library being defined.

```
DESCRIPTION 'Template Program'
```

EXETYPE Statement

Syntax: `EXETYPE [OS2 | WINDOWS | UNKNOWN]`

This statement specifies under which operating system the application (or dynamic-link library) is to run. This statement is optional and provides an additional degree of protection against the program being run in an incorrect operating system.

The EXETYPE keyword can be followed by a descriptor of the operating system:

OS2	OS/2 applications and dynamic-link libraries (default)
WINDOWS	Windows** applications
UNKNOWN	Other applications

The effect of EXETYPE is to set bits in the header that identify operating-system type. Operating-system loaders can check these bits.

EXPORTS Statement

Syntax:

```
EXPORTS
      exportdefinitions
```

This statement defines the names and attributes of the functions exported to other modules and of the functions that run with I/O privilege.

Note: The term *export* refers to the process of making a function available to other run-time modules. By default, functions are hidden from other modules at run time.

Normally, the EXPORTS statement is meaningful only for functions within dynamic-link libraries and for functions that execute with I/O privilege.

The EXPORTS keyword marks the beginning of the export definitions. Each definition is entered on a separate line. You need to give an export definition for each dynamic-link routine you want to make available to other modules.

Export-Definition Syntax

```
entryname [=internalname] [@ord[RESIDENTNAME]] [pwords]
```

<entryname>
: The function name as it is known to other modules.

<internalname>
: The actual name of the export function as it appears within the module itself; by default, this name is the same as <entryname>.

<ord>
: The function's ordinal position within the module definition table. If this field is used, the function's entry point can be invoked by name or by ordinal. Use of ordinal positions is faster and may save space.

RESIDENTNAME
: Indicates that the function's name be kept resident in memory at all times. This keyword is applicable only if <ord> is used. If <ord> is not used, OS/2 automatically keeps the names of all exported functions resident in memory by default.

<pwords>
: The total size of the function's parameters, as measured in words (bytes divided by two). This field is required only if the function executes with I/O privilege. When a function with I/O privilege is called, OS/2 consults <pwords> to determine how many words to copy from the caller's stack to the I/O-privileged function's stack.

Example

```
EXPORTS
  SampleRead = read2bin @8
  StringIn = str1       @4 RESIDENTNAME
  CharTest  6
```

This example defines three export functions:

- SampleRead
- StringIn
- CharTest

The first two functions can be accessed either by their exported names or by an ordinal number. Note that in the module's own source code, these functions are actually defined as read2bin and str1, respectively. The last function runs with I/O privilege and therefore is given with the total size of the parameters - six words.

IMPORTS Statement

Syntax:

```
IMPORTS
  importdefinitions
```

This statement defines the names of the functions imported for the application or library.

Note: The term *import* refers to the process of declaring that a symbol is defined in another run-time module (a dynamic-link library).

Typically, LINK386 uses an import library to resolve external references to dynamic-link symbols. However, the IMPORTS statement provides an alternative for resolving these references within a module.

The IMPORTS keyword marks the beginning of the import definitions. This keyword is followed by one or more import definitions, each on a separate line. Each import definition corresponds to a particular function.

Import-Definition Syntax

```
[internalname=]modulename.entry
```

<internalname>
: The name that the importing module uses to call the function. Thus, <internalname> appears in the source code of the importing module, although the function can have a different name in the module where it is defined. By default, internalname is the same as <entry>.

<modulename>
: The name of the application or library that contains the function.

<entry>
: The function to be imported; can be a name or an ordinal value. (Ordinal values are set in an EXPORTS statement.) If an ordinal value is given, then <internalname> is required.

Note: A given function has a name for each of three different contexts. The function has a name used by the exporting module (where it is defined), a name used as an entry point between modules, and a name as it is used by the importing module (where it is called). If neither module uses the optional <internalname> field, the function has the same name in all three contexts. If either of the modules uses the <internalname> field, the function may have more than one distinct name.

Syntax:

```
IMPORTS
  Sample.SampleRead
  SampleA.SampleWrite
  ReadChar = Read.1
```

This example defines three functions to be imported:

- SampleRead
- SampleWrite
- A function that has been assigned an ordinal value of 1

The functions are found in the modules Sample, SampleA, and Read, respectively. The function from the Read module is referred to as ReadChar in the importing module. The original name of the function, as it is defined in the Read module, may or may not be known.

HEAPSIZE Statement

Syntax: `HEAPSIZE bytes | MAXVAL`

This statement defines the size of the application's local heap in bytes. This value affects the size of the automatic data segment.

The field <bytes> contains any positive integer. You can enter <bytes> in decimal, octal, or hexadecimal radix. See Entering Numeric Arguments.

Instead of entering a number for <bytes>, you can enter the keyword MAXVAL. This sets the heap size such that the default data segment DGROUP is exactly 64K. MAXVAL is useful in bound applications in which you want to force a 64K requirement for DGROUP.

Example

```
HEAPSIZE 4000
```

This example sets the local heap to 4,000 bytes.

LIBRARY Statement

Syntax: `LIBRARY [libraryname][initialization] [termination]`

This statement identifies the executable file as a dynamic-link library and optionally defines the name and library module initialization required.

If <libraryname> is given, it becomes the name of the library as it is known by OS/2. This name can be any valid file name. If <libraryname> is not given, the name of the executable file - with the extension removed - becomes the name of the library.

If <initialization> is given, it defines the library initialization required and can be one of the values below. If omitted, <initialization> defaults to INITGLOBAL.

INITGLOBAL
: The library initialization routine is called only when the library module is initially loaded into memory.

  Using this keyword without a termination flag implies TERMGLOBAL for DLLs with 32-bit entry points.

INITINSTANCE
: The library initialization routine is called each time a new process gains access to the library.

  Using this keyword without a termination flag implies TERMINSTANCE for DLLs with 32-bit entry points.

If <termination> is given, it defines the library termination required and can be one of the values below. If omitted, <initialization> defaults to TERMGLOBAL. The termination flag can only apply to DLLs with 32-bit entry points.

TERMGLOBAL
: The library termination routine is called only when the library module is unloaded from memory.

  Using this keyword without an initialization flag implies INITGLOBAL.

TERMINSTANCE
: The library termination routine is called each time a process relinquishes access to the library.

  Using this keyword without an initialization flag implies INITINSTANCE.

If the LIBRARY statement is included in a module definition file, the NAME statement cannot appear. If no LIBRARY statement appears, the module definition file describes an application.

The following example assigns the name *calendar* to the dynamic-link library and specifies that library initialization be performed each time a new process gains access.

```
LIBRARY calendar INITINSTANCE
```

NAME Statement

Syntax: `NAME [appname][apptype]`

This statement identifies the executable file as an application and optionally defines the name and type.

If <appname> is given, it becomes the name of the application as it is known by OS/2. This name can be any valid file name. If <appname> is not given, the name of the executable file - with the extension removed - becomes the name of the application.

If <apptype> is given, it defines the type of application:

WINDOWAPI
: Presentation Manager* application. The application uses the API provided by the Presentation Manager and must be executed in the Presentation Manager environment.

WINDOWCOMPAT
: Application compatible with Presentation Manager. The application can run inside the Presentation Manager, or it can run in a

separate screen group. An application can be of this type if it uses the proper subset of OS/2 video, keyboard, and mouse functions supported in the Presentation Manager applications.

NOTWINDOWCOMPAT
: Application that is not compatible with the Presentation Manager and must operate in a separate screen group from the Presentation Manager.

If the NAME statement appears, the LIBRARY, VIRTUAL DEVICE and PHYSICAL DEVICE statements cannot appear. If none of these statements appear, the module definition file is assumed to describe an application.

Example

The following example assigns the name *calendar* to the application being defined.

```
NAME calendar WINDOWCOMPAT
```

This application is Presentation Manager-compatible.

OLD Statement

Syntax: `OLD 'filename'`

This statement directs LINK386 to search another dynamic-link module for export ordinals. Exported names in the current module that match exported names in the OLD module are assigned ordinal values from that module unless one of the following conditions is in effect:

- The name in the OLD module has no ordinal value assigned.
- An ordinal value is explicitly assigned in the current module.

This statement is useful for preserving export ordinal values throughout successive versions of a dynamic-link module. The OLD statement has no effect on application modules.

PHYSICAL DEVICE Statement

Syntax: `PHYSICAL DEVICE [drivername]`

This statement identifies the executable file as a physical device driver.

If <drivername> is given, it becomes the name of the driver as it is known by OS/2. This name can be any valid file name. If <drivername> is not given, the name of the executable file - with the extension removed - becomes the name of the driver.

PROTMODE Statement

Syntax: `PROTMODE`

This statement specifies that the module runs only in protected mode and not in Windows or dual mode. This statement is always optional and permits a protected mode only application to omit some information from the executable file header.

If this statement is not included in the module definition file, LINK386 assumes the application can be run in either real or protected mode.

STACKSIZE Statement

Syntax: `STACKSIZE number`

This statement performs the same function as the LINK386 /ST option; if both are used, the STACKSIZE statement overrides the /ST option. See Control Stack Size (/ST).

The field <number> contains a positive integer. You can specify <number> in decimal, octal, or hexadecimal radix. See Entering Numeric Arguments.

Example

```
STACKSIZE 4096
```

This example allocates 4,096 bytes of local-stack space.

STUB Statement

Syntax: `STUB 'filename'`

This statement adds a DOS executable file to the beginning of the application or library being created. The stub is invoked whenever the module is executed under DOS. Typically, the stub displays a message and terminates execution. By default, LINK386 adds its own standard stub for this purpose.

The field <filename> specifies the DOS executable file to be added. LINK386 looks for <filename> in the current directory and in the directories specified by the PATH environment variable.

Example

```
STUB 'STOPIT.EXE'
```

This example appends the DOS executable file STOPIT.EXE to the beginning of the module. STOPIT.EXE is executed when the module is run under DOS.

VIRTUAL DEVICE Statement

Syntax: `VIRTUAL DEVICE [drivername]`

This statement identifies the executable file as a virtual device driver.

If <drivername> is given, it becomes the name of the driver as it is known by OS/2. This name can be any valid file name. If <drivername> is not given, the name of the executable file - with the extension removed - becomes the name of the driver.

MAP File to SYM File Creator (MAPSYM)

The MAPSYM program creates .SYM files from .MAP files. .SYM files are used by the kernel debugger for symbolic debugging.

Note: You must run MAPSYM from the directory in which the file to be converted is located.

To create a .SYM file, follow these steps:

1. Make sure you are in the correct directory.
2. At the prompt type the following:

```
mapsym filename
```

Note that the .MAP extension is not required.

Help

To display MAPSYM help, type *MAPSYM* at the prompt, with no arguments. The appropriate copyright statement appears, along with the following:
usage: mapsym [-aln] mapfile

Options

The following options may be used with MAPSYM:

/A	Omits Alphabetical sorting of symbols.
/N	Includes source code line Numbers in *.SYM file.
/L	Produces verbose Listing.

View and Set Program Type For Executable File (MARKEXE)

The MARKEXE program enables you to view and set the program type for an executable file. The program type identifies the OS/2 sessions in which a program can run.

For applications running on OS/2 for SMP Version 3, MARKEXE enables you to set the MPUNSAFE bit, which forces the application to always run in uniprocessor mode. See Requirements for Multi-Processing for information on when to set the MPUNSAFE bit.

Use MARKEXE with the OS/2 Linear Executable Linker (LINK386) or the OS/2 Segmented Executable Linker (LINK) to change or set the program type of programs you have created and to set or unset the MPUNSAFE bit.

You can set DLL initialization and termination and also enable long file name support. When using LINK386, you can set DLL initialization and termination; long file name support is already set. When using LINK, you can set DLL initialization and long file name support.

Command-Line Syntax

MARKEXE uses the following syntax:

```
MARKEXE [/?] [FORCE] [NO] [option] filename...
```

Filename is a file name or a list of file names. Global file-name characters (*.EXE) also can be used. For descriptions of the above terms, see Syntax Definitions. If no option is given, DISPLAY is assumed.

Typing MARKEXE /? at the command line displays the appropriate copyright statement along with a list of options.

```
DISPLAY         - display status of flags
DLLINIT         - per-process initialization
DLLTERM         - per-process termination
WINDOWAPI       - window api (PM application)
WINDOWCOMPAT    - window compatible application
NOTWINDOWCOMPAT - not window compatible application
UNSPECIFIED     - unspecified application type
LFNS            - long file name support
MPUNSAFE        - multi-processor unsafe application
SETVERSION      - Write 32 bit Module Version Field
```

Syntax Definitions

MARKEXE has the following keywords, options, and program types. You can also specify any number of files to be viewed or marked.

KEYWORDS

FORCE	Marks the executable file with OS/2 as the target operating system even though the file was marked for another operating system. Using FORCE might produce internally inconsistent executable files.
NO	Sets the command to the opposite condition. This keyword does not apply to the DISPLAY option or any of the Program Type options.

OPTIONS

DISPLAY	Displays the application type in a message; does not make any changes. (This is the default option.)
DLLINIT	Sets per process initialization for the dynamic link library.
DLLTERM	Sets per process termination for the dynamic link library. (Use with LINK386 only.)
LFNS	Enables support of long file names. (Use with LINK only.)
MPUNSAFE	Marks the application as unsafe for running in a multi-processor environment.
SETVERSION	Extracts vendor-specific values from text strings embedded into the executable module, converts them into bitfields, and stores the result into a 32-bit "Module Version" field in the executable header. This field can be displayed with the EXEHDR utility. MARKEXE searches for version strings in the following sections of the executable module: • The description field (established at link-time by a DESCRIPTION statement in the module definition file). • The .ENHVERSION extended attribute, if present. • Throughout the body of the executable module.

Note: Only one option may be entered on the command line.

PROGRAM TYPES

MARKEXE does not modify the file if the executable file's program type is the same as the requested type. It displays a message instead.

WINDOWAPI	The application is a Presentation Manager application and can run in the Presentation Manager session only.
WINDOWCOMPAT	The application can run in a Presentation Manager window or in an full-screen session.
NOTWINDOWCOMPAT	The application must run in an OS/2 full-screen session.
UNSPECIFIED	The application type is not known. By default, the OS/2 operating system will force the program to run in a full-screen session.

Note: Specifying an incorrect program type might cause undesirable results when you try to run that program. For example, do not change a WINDOWCOMPAT program to WINDOWAPI.

Only one program type may be entered on the command line.

Viewing Program Type

To display the program type of an executable file without changing the file, specify only a file name, omitting an option.

```
MARKEXE filename.exe
```

Example

To view the program type of MYPROG.EXE, type the following:

```
MARKEXE myprog.exe
```

MARKEXE displays the type in a message that looks like this:

```
myprog.exe: OS/2 1.x, WINDOWCOMPAT, LFNS
```

Setting Program Type

To set the program type of an executable file, specify one of the program types. More than one executable file can be set to the same program type on a single command line.

```
MARKEXE type filename.exe another.exe
```

Examples

To set WINDOWCOMPAT as the program type of MYPROG.EXE, type:

```
MARKEXE WINDOWCOMPAT myprog.exe
```

To set WINDOWAPI as the program type of several executable files, type:

```
MARKEXE WINDOWAPI marion.exe alex.exe
```

Requirements for Multi-Processing

The following compatibility's requirements should be considered before running an application in multi-processor mode:

- An application or associated subsystem must not use the 'INC' instruction as a semaphore without prepending a 'LOCK' prefix. On a UniProcessor (UP) system this instruction can be used as a high-performance semaphore without calling any other operating system service if the semaphore is free and when the semaphore is clear and there are no waiters for the semaphore. Because the INC instruction cannot be interrupted once started, and because the results would be stored in the flags register which are per thread, then it could be used safely as a semaphore.

In an OS/2 for SMP environment, this technique will not work because it is possible that two or more threads could be executing the same INC instruction, receiving the same results in each processor's or thread's flag register and thinking that they have the semaphore.

- A 486 or greater instruction such as CMPXCHG has the same problem as above if a 'LOCK' prefix is not prepended before the instruction.

- An application or associated subsystem which relies on priorities to guarantee execution of it's threads within a process will not work in OS/2 for SMP. For example, an application may have a time-critical and an idle thread, and may assume that while the time-critical thread is executing the idle thread will not get any execution time unless the time-critical thread explicitly yields the CPU. In an OS/2 for SMP environment it is possible that both the time-critical and the idle threads are executing simultaneously on different processors.

The above compatibility requirements apply only to multi-threaded applications, and therefore do no apply to DOS and WINOS2 applications. However, you are strongly encouraged to write 32-bit multi-threaded applications for better performance and portability on OS/2 for SMP.

Given the possibility that some set of applications may use one of these techniques, OS/2 for SMP Version 3 provides a mechanism whereby these multi-threaded applications can execute in UP mode. Only one thread of that process would be allowed to execute at any given time. That thread could execute on any one of the processors. MARKEXE can be used to mark the EXE file as uniprocessor only. OS/2 forces the process to run in the uniprocessor mode when the loader detects that the EXE file has been marked as uniprocessor only.

Note: OS/2 Warp Version 3 (non-SMP) still ships the EXECMODE utility.

Preprocess Message Source File Utility (MKCATDEF)

MKCATDEF preprocesses a message source file for input to GENCAT (see Generate Message Catalog Utility (GENCAT)).

MKCATDEF reads a message source file containing symbolic identifiers and produces the following output:

- The SYMBOLNAME.H file, containing statements that equate symbolic identifiers with the set numbers and message ID numbers assigned by MKCATDEF. You must include the SYMBOLNAME.H file in your application program to associate the symbolic names to the set and message numbers assigned by MKCATDEF.

- A new message source file containing message numbers instead of symbolic message identifiers. This output is suitable for input to GENCAT.

MKCATDEF sends the new message source file, with numbers instead of symbolic identifiers, to standard output. You can use MKCATDEF output as input to GENCAT in the following ways:

- Use MKCATDEF with a > (redirection symbol) to write the new message source to a file. Use this file as input to GENCAT.

- Pipe the MKCATDEF output file directly to GENCAT.

After running MKCATDEF, you can use symbolic names in an application to refer to messages.

Syntax

```
mkcatdef SymbolName SourceFile ...  [  -h ]
```

Flags

-h Suppresses the generation of a SYMBOLNAME.H file. This flag must be the last argument to MKCATDEF.

Examples

To process the SYMB.MSG message source file and redirect the output to the SYMB.SRC file, enter:

```
mkcatdef symb symb.msg > symb.src
```

The generated SYMB.H file looks similar to the following:

```
#ifdef        _H_SYMB_MSG
#define       _H_SYMB_MSG
#include <limits.h>
#include <nl_types.h>
#define       MF_SYMB "symb.cat"
/* The following was generated from symb.src. */
/* definitions for set MSFAC */
#define       SYM_FORM        1
#define       SYM_LEN 2
#define       MSG_H   6
#endif
```

MKCATDEF also creates the SYMB.SRC message catalog source file for the gencat command with numbers assigned to the symbolic identifiers:

```
$quote " Use double quotation marks to delimit message text
$delset 1
$set 1
1     "Symbolic identifiers can only contain alphanumeric \
characters or the _ (underscore character) but must begin with \
an alpha or _ character\n"
2     "Symbolic identifiers cannot be more than 64 \
characters long\n"
5     "You can mix symbolic identifiers and numbers\n"
$quote
6     remember to include the "msg_h" file in your program
```

The assigned message numbers are noncontiguous because the source file contained a specific number. MKCATDEF always assigns the previous number plus 1 to a symbolic identifier.

Note: MKCATDEF inserts a $delset command before a $set command in the output message source file. This means you cannot add, delete, or replace single messages in an existing catalog when piping to the gencat command. You must enter all messages in the set.

Related Information

- GENCAT (see Generate Message Catalog Utility (GENCAT))
- catclose, catgets and catopen (see *C Library Reference*)

Make Message File (MKMSGF)

The MKMSGF program reads the input message file that you specify and creates an output message file that DosGetMessage uses to display messages.

There are two ways that the output message file can be used:

- Selected messages can be bound to the message segment of an executable file using the MSGBIND program.
- Messages can be accessed directly from the output message file.

See How Message Retrieval Works for additional information.

Syntax

```
MKMSGF infile outfile [options]
```

OR

```
MKMSGF @controlfile
```

The infile field specifies the input file that contains message definitions. The input-file name can be any valid OS/2 file name, optionally preceded by a drive letter and a path.

The outfile field specifies the output file created by MKMSGF. The output-file name can be any valid OS/2 file name, optionally preceded by a drive letter and a path.

To differentiate between the two files, the following convention is recommended, using the same file name.

- The **infile** file should have a .TXT extension.
- The **outfile** file should have a .MSG extension.

Note: The output file ***cannot*** have the same file name and extension as the input file.

Help

There are two ways to display MKMSGF help.

Short Syntax Help

To display a short version of MKMSGF syntax help, type `MKMSGF` at the prompt, with no parameters. The following will be displayed:

```
MKMSGF infile[.ext] outfile[.ext] [/V]
[/D <DBCS range or country>] [/P <code page>]
[/L <language id,sub id>]
```

Long Help

To display a longer version of MKMSGF help, including defaults, country codes, and language IDs, type `MKMSGF /?` at the prompt. The following will be displayed:

Use MKMSGF as follows:

```
MKMSGF <inputfile> <outputfile> [/V]
        [/D <DBCS range or country>]
        [/P <code page>]
        [/L <language family id,sub id>]
```

where the default values are:
   code page - none
   DBCS range - none
A valid DBCS range is: n10,n11,n20,n21,...,nn0,nn1
A single number is taken as a DBCS country code.
For a complete list of code pages and country codes,
see the code page table under COUNTRYCODE in the online book
Control Program Programming Guide and Reference.
Supported OS/2 language/sublanguage ID values include:

Code	Family	Sub	Language	Principal country
ARA	1	2	Arabic	Arab Countries
BGR	2	1	Bulgarian	Bulgaria
CAT	3	1	Catalan	Spain
CHT	4	1	Traditional Chinese	R.O.C.
CHS	4	2	Simplified Chinese	P.R.C.
CSY	5	1	Czech	Czechoslovakia
DAN	6	1	Danish	Denmark
DEU	7	1	German	Germany
DES	7	2	Swiss German	Switzerland
EEL	8	1	Greek	Greece
ENU	9	1	US English	United States
ENG	9	2	UK English	United Kingdom
ESP	10	1	Castilian Spanish	Spain
ESM	10	2	Mexican Spanish	Mexico
FIN	11	1	Finnish	Finland
FRA	12	1	French	France
FRB	12	2	Belgian French	Belgium
FRC	12	3	Canadian French	Canada
FRS	12	4	Swiss French	Switzerland
HEB	13	1	Hebrew	Israel
HUN	14	1	Hungarian	Hungary
ISL	15	1	Icelandic	Iceland
ITA	16	1	Italian	Italy
ITS	16	2	Swiss Italian	Switzerland
JPN	17	1	Japanese	Japan
KOR	18	1	Korean	Korea
NLD	19	1	Dutch	Netherlands
NLB	19	2	Belgian Dutch	Belgium

```
NOR     20       1   Norwegian - Bokmal      Norway
NON     20       2   Norwegian - Nynorsk     Norway
PLK     21       1   Polish                  Poland
PTB     22       1   Brazilian Portuguese    Brazil
PTG     22       2   Portuguese              Portugal
RMS     23       1   Rhaeto-Romanic          Switzerland
ROM     24       1   Romanian                Romania
RUS     25       1   Russian                 U.S.S.R.
SHL     26       1   Croato-Serbian (Lat     Yugoslavia
SHC     26       2   Serbo-Croatian (Cyr     Yugoslavia
SKY     27       1   Slovakian               Czechoslovakia
SQI     28       1   Albanian                Albania
SVE     29       1   Swedish                 Sweden
THA     30       1   Thai                    Thailand
TRK     31       1   Turkish                 Turkey
URD     32       1   Urdu                    Pakistan
BAH     33       1   Bahasa                  Indonesia
```

Input Message File

The input message file is a standard ASCII file that contains three types of lines:

- Comment lines
- Component identifier line
- Component message lines

Comment Lines

Comment lines are allowed anywhere in the input message file, except between the component identifier and the first message. Comment lines must begin with a semicolon (;) in the first column.

In the Input Message File Example, the comment lines are

```
; This is a sample of an input
; message file for component DOS
; starting with three comment lines.
```

Component Identifier Line

The component-identifier line contains a three-character name identifier that precedes all MKMSGF message numbers.

In the example, the component identifier is DOS.

Component-Message Lines

Each component-message line consists of a message header and an ASCII text message.

The message header is comprised of the following parts:

- A three-character component identifier
- A four-digit message number
- A single character specifying message type (`E, H, I, P, W, ?`)
- A colon (:)
- Followed by a blank space.

The following message types are used:

Type	Meaning
E	Error
H	Help
I	Information
P	Prompt
W	Warning
?	no message assigned to this number

The message header must begin in the first column of the line. Only one header can precede the text of a message, although a message can span multiple lines.

Message numbers can start at any number, but messages must be numbered sequentially. If you do not use a message number, you must insert an empty entry in its place in the text file. An empty entry consists of the message number, with `?` as the message type, and no text.

The character % has a special meaning when used within the text of a message:

%0 is placed at the end of a prompt (type P) to prevent DosGetMessage from executing a carriage return and line feed. This allows the user to be prompted for input on the same line as the message text.

%1 - %9 are used to identify variable string insertion within the text of a message. These variables correspond to the Itable and IvCount parameters in the DosGetMessage call.

Component-Message Example

For example, `DOS0100E:` is DOS error message 100. For additional examples, see the Input Message File Example.

Output File

The output file contains the indexed message file that DosGetMessage will use. The output-file name can be any valid OS/2 file name, optionally preceded by a drive letter and a path. The output file ***cannot*** have the same name as the input file.

To differentiate between the two files, the following convention is recommended, using the same file name.

- The **infile** file should have a .TXT extension.
- The **outfile** file should have a .MSG extension.

Help-message file names begin with the component identifier, followed by H.MSG. For example, the help file associated with the component identifier DOS would be DOSH.MSG.

Options

Text-based messages in different code pages can be created using MKMSGF to display errors, help information, prompt, or provide general information to the application user.

MKMSGF uses the following parameter formats to build message files:

```
MKMSGF infile outfile /Pcodepage

MKMSGF infile outfile /Ddbcsrange or country id
```

```
MKMSGF infile outfile /LlangID,VerId

MKMSGF infile outfile /V

MKMSGF infile outfile /?

MKMSGF @controlfile
```

- Infile is the ASCII-text source file.

  Example:

  ```
  MSG
  MSG0001I: (mm%4dd%4yy) %2%4%1%4%3
  MSG0002I: (dd%4mm%4yy) %1%4%2%4%3
  MSG0003I: Current date is: %0
  ```

  %0 is a special argument that displays a prompt for user input.

  %1 - %9 are the arguments the user can use to insert text in a message.

- Outfile is the binary output message file.
- @controlfile is the message definition file.

Options

/P	Code-page ID for the input message file. See /P Option
/D	DbcsRange or country ID for the input message file. See /D Option
/L	Language family ID (one word) and language version ID (one word). See /L Option
/V	Verbose display of message file control variables as the message file is being created. See /Verbose Option Output Example
/?	Help display of command syntax for MKMSGF.

Note: Any combination of /P, /D, /L, and /V switches can be used for either the command line or @controlfile execution method.

The / switch prefix and the - prefix can be used interchangeably when defining switches to MKMSGF.

/Verbose Option Output Example

Following is a sample of MKMSGF output, using the Verbose option (/V). This output was produced using the following command:

```
mkmsgf myapp.txt myapp.msg /v
```

```
  strIn     = myapp.txt
  strOut    = myapp.msg
  StrIncDir = (null)
  CodePages = 437
  Language family id = 0 and sub id = 0
  Language family id and sub id = unspecified
  flags     = none
  CP_type   = SBCS
"myapp.txt": length = 382 bytes.
29 messages scanned.  Writing output file...
Size of table entry: word
```

/P Option

The Code-page option (/P) specifies the code-page ID for that input message file.

For a complete list of code pages, see the code page table under COUNTRYCODE in the online book *Control Program Programming Reference* .

Up to 16 /P combinations can be saved with the message file.

/P cannot be used to identify DBCS data.

/D Option

The DBCS option (/D) specifies the DBCS Range or country ID for that input message file.

A valid DBCS country ID will cause the initialization of valid DBCS ranges to be set up for this file.

See DBCS Code Pages and Country Codes for valid DBCS country codes.

/L Option

The Language option (/L) specifies the language family ID (one word) and language version ID (one word).

Valid combination of language family and language version will be set for this file.

A valid language family with invalid or undefined language version id will cause a default value of 1 to be set for language version.

Control Files

The control file (@controlfile) is used to create multiple-code-page message files. The at sign (@) is not part of the file name, but rather, a delimiter required before a control-file name.

The control file has the following format:

Example:

```
root.in root.out /Pcodepage /Ddbcsrang/ctryid /LlangID,VerId
sub.001 sub1.out /Pcodepage /Ddbcsrang/ctryid /LlangID,VerId
                 .
                 .
sub.00n subn.out /Pcodepage /Ddbcsrang/ctryid /LlangID,VerId
```

The help option (/?) is invalid due to the purpose of the definition file.

Note: Any combination of /P /D /L and /V switches can be used for either the command line or msg_definition_file execution method.

Input Message File Example

Following is an example of an input message file:

```
; This is a sample of an input
; message file for component MAB
; starting with three comment lines.
MAB
MAB0100E: File not found
MAB0101?:
MAB0102H: Usage: del [drive:][path] filename
MAB0103?:
MAB0104I: %1 files copied
MAB0105W: Warning! All data will be destroyed!
MAB0106?:
MAB0107?:
MAB0108P: Do you wish to apply these patches (Y or N)? %0
MAB0109E: Divide overflow
```

Make Template File (MKTMPF)

MKTMPF creates template repository files from text input files.

A template repository (also referred to as a *repository file* or *repository* in this document) is a binary file used by the operating system's Error Logging Facility to find error descriptions, causes, and actions in various message files.

The first step in creating a repository file is to create a text input file for MKTMPF using any text editor. The contents and format of these files are described later in this document. These input files can have any file name that is valid for your file system. A standard extension, TMP, is recommended, but not required.

MKTMPF reads and validates this file, reporting any errors or warnings, before translating it to binary data and creating a repository file.

Starting MKTMPF

MKTMPF is started from the command line, so it can be used from batch files and *makefiles* in a batch environment.

Command-Line Syntax

The command-line syntax for MKTMPF is as follows:

```
mktmpf [-?][-c][-q][-v][-O <outputfile>][-W <level>][-V<text>]<input file>
```

Command-Line Switches and Options

There are two types of command-line arguments: switches and options. Each command-line argument is denoted by a character prefixed by a hyphen (-) or a slash (/), and sometimes followed with a value to be used by the program. Command-line arguments are case-sensitive.

Switches are used to set the state of a flag that affects the program's behavior. In effect, the presence of a given switch on the command line sets a flag that has a default value of FALSE, to TRUE. Switches consist of a hyphen or slash followed by a lowercase alphabetic character. They do not take a value, and can be grouped together with a single hyphen or slash (for example, -cq).

Options are used to set the value of one of the variables that the program uses. They differ from switches in that they require at least one value to be entered. They consist of a hyphen or slash followed by an uppercase alphabetic character.

The POSIX command-line delimiter "--" must be used when needed to separate options from operands. Any option-argument values that begin with a hyphen, plus sign, comma, at sign (@), space, or any special character must be enclosed in double quotation marks. If any valid options are entered more than once on the command line, the last entry will be used and all previous occurrences will be ignored.

Note:

Options and switches are case-sensitive.

Options can be specified in any order.

<arg> indicates that "arg" is mandatory.

[arg] indicates that "arg" is optional.

If the -O, -V, or -W option immediately precedes the *<input file>* parameter, the two must be separated with "--". For example:

```
-O repository.rep -- infile.tmp
```

The following are the switches and options that MKTMPF supports:

-?	A switch that displays help for MKTMPF.
-c	A switch that checks the input for syntax errors and *does not* create the repository file.
-q	A switch that suppresses the banner that MKTMPF displays when it starts.
-v	A switch that turns on verbose mode for logging and reporting purposes.
-O <output file>	An option that specifies the name of the *output repository* file. If this option is not provided, the output file name will match the input file name except the extension will be .REP. -O also overrides the repository path name specified in the input file.
-V <text>	An option that allows the user to specify the version of the repository file. The version is 16 characters of user-defined text.
-W <level>	An option that specifies the warning level to be used. Valid warning levels are: 1 - Displays fatal errors only. 2 - Displays fatal and severe errors. 3 - Displays fatal errors, severe errors, and warnings. 4 - Displays all message types above, and informational messages. The default warning level is 3.
<input file>	The name of the file that contains the input statements used to build the templates. It must always be specified as the final parameter.

Input Files

The input file used by MKTMPF to create repository files can be created with a text editor (or other method of the user's choosing). It should contain one set of default path names and a set of one or more template entries. The following is an example of an input file with a single template.

Note:

The keywords surrounded by <> are replaced by the appropriate data.

Blank lines are ignored.

Any line in which an asterisk appears before any other non-white space is ignored.

Keywords are not case-sensitive; key values are case-sensitive.

`Descriptive_name` and `comment` must be in double quotation marks.

```
************************************************
*** This section appears once in the input file   ***
************************************************
 Repository_pathname = xxxxxxxxxx
 Default_message_pathname = xxxxxxxxxx
 Default_causes_pathname = xxxxxxxxxx
 Default_actions_pathname = xxxxxxxxxx
 Default_details_pathname = xxxxxxxxxx
 Descriptive_name = "xxxxxxxxxx"
************************************************
*** The remaining lines define a single template. ***
*** Additional templates may be appended by       ***
*** replicating the following text.               ***
************************************************
 <action> Template_number = xxxxxxxx
 Comment = "xxxxxxxxxxxxxxxx"
 Message_number = xxxxxxxx
 Log_Class = x
 Message_pathname = xxxxxxxxxxxx
 Causes_pathname = xxxxxxxxxxxxxx
 Actions_pathname = xxxxxxxxxxxx
 Details_pathname = xxxxxxxxxxxxxxxxxx
 Fail_causes = xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx
 Fail_actions = xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx
 Install_causes = xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx
 Install_actions = xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx
 User_causes = xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx
 User_actions = xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx
 Detail_data = <length>,<offset>,<heading>,<type>
```

Note: Long file names, white space, and special characters are acceptable if the user's file system supports them. If a path name contains leading or trailing white space, it should be enclosed in double quotation marks. You can provide either a full path name or just the file name. If the file name is provided, it is the developer's responsibility to make sure it is in a directory specified in the DPATH of the system where the repository will be used.

Repository Keywords

Repository keywords appear once and pertain to the entire input file. If a repository keyword is specified more than once, the last occurrence of the keyword will be used.

Repository_pathname (required)
The name of the repository file where these templates are to be placed. This is a one-time parameter and, by convention, should be at the top of the input file (before any templates). The file extension, by convention, should be .REP and you can provide either a full path name or just the filename. If a path name contains leading or trailing white space, it should be enclosed in double quotation marks. If only the file name is provided, it will be placed in the current directory. If you do not specify a file name, one will be provided.

Default_message_pathname (optional)
Default error-message file name that will be used if the template definition does not provide one. This is a one-time parameter and, by convention, should be at the top of the input file (before any templates are defined).

Default_causes_pathname (optional)
Default causes-message file name that will be used if the template definition does not provide one. This is a one-time parameter and, by convention, should be at the top of the input file

(before any templates are defined).

Default_actions_pathname (optional) — Default actions-message file name that will be used if the template definition does not provide one. This is a one-time parameter and, by convention, should be at the top of the input file (before any templates are defined).

Default_details_pathname (optional) — Default details-message file name that will be used if the template definition does not provide one. This is a one-time parameter and, by convention, should be at the top of the input file (before any templates are defined).

Descriptive_name (optional) — A 256-character (including ") descriptive name used to identify the contents of the template file. This is a one-time parameter and must be at the top of the input file.

Template Keywords

Template keywords appear once for each template in this repository input file. If a template keyword is specified more than once, the last occurrence of the keyword will be used.

<action> (optional) — Action that will be performed against this template entry. **This keyword prefixes the *Template_number* keyword.** If omitted, "ADD" will be the action taken. Valid actions are:

ADD	Add this template to the specified repository.
COMMENT	Treat this entry as a comment only and *do not change* the specified repository.

Template_number (required) — The unique number that is to be associated with this template. MKTMPF will validate the number for uniqueness. This number is *required* for each template in the input file, and it cannot be 0.

Note: This signifies the start of a template entry. The end of a template is detected by another *Template_number* entry or the end of file.

Comment (optional) — A 40-character (including ") user comment to be included with this template that might help the user identify what the template is used for. A sample comment would be:

"I/O error on printer LPT2," which identifies that this template is to be used by FFSTProbes that detect I/O errors on LPT2.

Message_number (required) — The main message number for this error. The message will be found in the message file specified by *Message_pathname* or *Default_message_pathname* . This number is *required* for each template in the input file. A message number equal to 0 will be considered an error, because this could indicate a missing message number.

Log_Class (optional) — The type (*Log_Class*) of this error; H = Hardware, S = Software (default).

Message_pathname (optional) — Name of the message file that contains the main message for this template (Message = xxx).

Note: If this parameter is not provided, the Default_message_pathname information will be used.

Causes_pathname (optional) — Name of the message file containing messages that provide cause information.

Note: If this parameter is not provided, the Default_causes_pathname information will be used.

Actions_pathname (optional) — Name of the message file containing messages that provide specific information about actions that might correct this error.

Note: If this parameter is not provided, the Default_actions_pathname information will be used.

Details_pathname (optional) — Name of the message file containing heading information for detail data formatting.

Note: If this parameter is not provided, the Default_details_pathname information will be used.

Fail_causes (optional)
Up to four message numbers, separated by commas, that can be found in the Causes message file. These messages will provide information relating to the failure or error that occurred.

Fail_actions (optional)
Up to four message numbers, separated by commas, that can be found in the Actions message file. These messages will provide information relating to the actions that should be taken to either correct or gather more information about the error.

Install_causes (optional)
Up to four message numbers, separated by commas, that can be found in the Causes message file. These messages will provide information relating to possible installation causes of the failure or error that occurred.

Install_actions (optional)
Up to four message numbers, separated by commas, that can be found in the Actions message file. These messages will provide information relating to the actions that should be taken to either correct or gather more information about the error.

User_causes (optional)
Up to four message numbers, separated by commas, that can be found in the Causes message file. These messages will provide additional user information relating to the failure or error that occurred.

User_actions (optional)
Up to four message numbers, separated by commas, that can be found in the Actions message file. These messages will provide information relating to the actions that should be taken to either correct or gather more information about the error.

Detail_data (optional)
Entries used to format the *user_data* portion of template records. There can be multiple entries. Each entry contains four pieces of information separated by commas. The information must be in the following order:

1. length
The length of data formatted by this instruction.

2. offset
The offset into the *user_data* area where this formatting record is to start. The first byte of *user_data* is position 0.

3. heading
The message number of the details file that is to be displayed before this data. A blank specifies that no heading will be displayed.

4. type
The type of formatting that is to be done:

0	The data is not to be used.
1	The data conforms to ISO 8859.1 and is to be displayed in ASCII characters.
2	The data is in binary format and should be converted to decimal before it is displayed.
3	Display the data in hexadecimal.
4	The data is Unicode and is to be displayed in ASCII characters.

The following table shows which entries are required for the specific actions of MKTMPF. If an entry is marked as "Optional," neither the entry type nor its value is required. You can enter the entry type with no value if you desire to have a placeholder.

Entry Type	Add	Comment
Repository_pathname	Required	Required
Default_Message_Pathname	Optional*	Optional*
Default_Causes_Pathname	Optional*	Optional*
Default_Actions_Pathname	Optional*	Optional*
Default_Details_Pathname	Optional*	Optional
<action>	Optional	Required
Template_number	Required	Optional
Comment	Optional	Optional
Message_number	Required	Optional

Log_Class	Optional	Optional
Message_pathname	Optional*	Optional
Causes_pathname	Optional*	Optional
Actions_pathname	Optional*	Optional
Details_pathname	Optional*	Optional
Fail_causes	Optional	Optional
Fail_actions	Optional	Optional
Install_causes	Optional	Optional
Install_actions	Optional	Optional
User_causes	Optional	Optional
User_actions	Optional	Optional
Detail_data	Optional	Optional

* Either the default path name or the path name in each template is required if the template is using messages. For examples, see the following table.

Required	When Required
Default_message_pathname or Message_pathname	Always
Default_causes_pathname or Causes_pathname	When providing cause information
Default_actions_pathname or Actions_pathname	When providing action information
Default_details_pathname or Details_pathname	When providing detail data information

---

Output File and Error Reporting

The output of MKTMPF is the repository file.

Error Reporting

Errors and other information are shown on the display. The log information for the verbose option includes:

- The current settings for the following:
    - Quiet mode
    - Log file name
    - Output file name
    - Warning level
    - Check only
    - Version text
- Informational and error messages

Message Segment Binder (MSGBIND)

The MSGBIND program binds a message segment to an executable program. It does this by reading an input file that specifies the executable files to modify. For each executable file, MSGBIND specifies which message files to scan, and for each message file, it specifies which messages to include in the executable file. Although the resulting executable file will be larger, access to messages will be faster.

In the OS/2 operating system, message segment/objects are packed with other application code. If the size of the code segment/object and the bound messages exceeds 64KB, the following statement in the program definition file (.DEF) isolates the application code from the message statement/object:

16 Bit Applications
SEGMENTS '_MSGSEG' CLASS 'CODE'

32 Bit Applications
SEGMENTS '_MSGSEG32' CLASS 'CODE'

Syntax

The MSGBIND command line has the following form:

```
MSGBIND infile
```

The **infile** field specifies the input file that identifies the executable files, output message files, and message numbers that will be bound. The input-file name can be any valid OS/2 file name and can include an optional file name extension.

Input File

The input file contains the following three types of lines:

- > Executable file
- < Message file
- Message numbers.

Executable file

The File in which messages are to be bound is preceded by a greater-than symbol (>). The name of the file can be any valid OS/2 file name.

Two or more different executable files can be modified by the specifications found in one input file. MSGBIND continues to use this file until it encounters another greater-than symbol.

Message file

The message file to be read from is preceded by a less-than sign (<). You create this file by using the MKMSGF program. The name can be any valid OS/2 file name. All message numbers that follow it are located in the specified message file and are copied to the current output executable file. MSGBIND reads the message-number list until it encounters one of the following: the end of the input file, a new output specification, or a new input message file.

Message Numbers

The messages in the message file are listed below the message-file name. Only those message numbers that you specify will be added. You can also specify an asterisk (*) to indicate that all messages within the message file will be added. Message numbers must consist of a three-letter component identifier followed by a four-digit message number.

See Input Message File for a more detailed description of message numbers.

Multiple Code-Page Message Files

Multiple code-page message files can also be bound to an executable file, which enables a user to bind to an application messages for different countries.

The following example shows how three messages in two different languages can be bound to an executable file:

```
MSGBIND infile
```

where *infile* consists of the following:

```
>PROG1.EXE
<TEXTUS.MSG
MSG0001
MSG0002
MSG0003
<TEXTIT.MSG
MSG0001
MSG0002
MSG0003
```

where:

- PROG1.EXE is the executable file to be modified.
- TEXTUS.MSG is the file, created using MKMSGF, which contains messages in US English.
- TEXTIT.MSG is the file, created using MKMSGF, which contains the same messages translated into Italian.
- MSGnnnn defines the messages to be bound to the application:

MSG	Message component ID
0001	Message number

Help

To display MSGBIND help, type `MSGBIND` at the prompt, with no parameters. The following will be displayed:

```
usage: MSGBIND scriptfile
```

How Message Retrieval Works

When an application requests the message retriever for text associated with a message number, a test is made to determine if there is a bound message segment with this executable file. If true, each bound message segment is searched for a match with the current session's code-page number.

If a match is made, then the message number is searched for. If it is found, the message will be returned to the caller. Otherwise, the search of remaining bound message segments will continue.

If no match results from a search of all message segments, the message file on the disk is searched. DosGetMessage will access the message file under any of the following conditions:

- The message file is in the current directory.
- The message file is in the path specified in the DPATH environment variable (protect mode).
- The message file is in the path specified in the APPEND environment variable (real mode).
- The fully-qualified file name is specified in DosGetMessage.

Sample Input File

```
>c:\cmd.exe
<c:\os20\dosutil.msg
DOS0100
DOS0123
DOS0245
>c:\format.exe
<c:\os20\dosutil.msg
DOS0001
DOS0006
<c:\format.msg
FMT0001
FMT0002
<c:\myown.msg
*
```

The first line of a MSGBIND input file specifies that the executable file to modify is CMD.EXE. The messages DOS0100, DOS0123, and DOS0245 are read from the file DOSUTIL.MSG and added to the CMD.EXE file. The MSGBIND program then encounters an executable-file option for the FORMAT.EXE file. The messages DOS0001 and DOS0006 are read from DOSUTIL.MSG and added to FORMAT.EXE. Next, the messages FMT0001 and FMT0002 are read from the file FORMAT.MSG and added to FORMAT.EXE. Finally, because an asterisk is specified, all the messages are read from the file MYOWN.MSG and added to FORMAT.EXE.

The files DOSUTIL.MSG and FORMAT.MSG in this example are two output-message-file names from the MKMSGF program.

Program Maintenance Utility Program (NMAKE)

The Program Maintenance utility program, NMAKE, automates the process of updating project files. NMAKE compares the modification dates for one set of files (the target files) with those of another set of files (the dependent files). If any dependent files have changed more recently than the target files, NMAKE executes a series of commands to bring the targets up-to-date.

Why Use NMAKE?

The most common use of NMAKE is to automate the process of updating a project after you make a change to a source file. Large projects tend to have many source files. Often, only a few of your source files need to be compiled when you make a change. You set up a special text file called a "description" file, or a "makefile", that tells NMAKE:

- Which files depend on others
- Which commands, such as compile and link commands, need to be carried out to bring your program up-to-date

This use of NMAKE is only one example of its power. By building suitable description files, you can use NMAKE to

- Make backups
- Configure data files
- Run programs when data files are modified

Running NMAKE

Run NMAKE by typing NMAKE on the operating-system command line. Supply input to NMAKE by either of two methods:

- Enter the input directly on the command line.
- Put your input into a *command file* (a text file, also called a *response file*) and enter the file name on the command line.

Press CTRL+C at any time during an NMAKE run to return to the operating system.

Note: Under the OS/2 operating system, do not use the ampersand character (&) to combine the NMAKE command with the CD, CHDIR, or SET command.

Using the Command Line

When using NMAKE at the command line, keep the following in mind:

- All fields are optional.
- If the /F option (Specify Description File) is specified, NMAKE uses the description file provided with the /F option. Otherwise, NMAKE will look in the current directory for a description file named MAKEFILE.

Command-Line Syntax

```
NMAKE [options] [macrodefinitions] [targets] [/F filename]
```

<options>
Specifies options that modify NMAKE's actions.

<macrodefinitions>
Lists macro definitions for NMAKE to use. Macro definitions that contain spaces must be enclosed by double quotation marks.

<targets>
Specifies the names of one or more target files to build. If you do not list any targets, NMAKE builds the first target in the description file.

/F <filename>

Gives the name of the description file where you specify file dependencies and which commands to execute when a file is out-of-date.

The following example:

```
NMAKE /S "program = flash" SORT.EXE SEARCH.EXE
```

- Invokes NMAKE with the /S option
- Defines a macro, assigning the string "flash" to the macro "program"
- Specifies two targets: SORT.EXE and SEARCH.EXE

By default, NMAKE uses the file named MAKEFILE as the description file.

Command-Line Help

To display NMAKE help, type `NMAKE /?` at the prompt. The appropriate copyright statement appears, along with the following:

```
Usage:
        NMAKE @commandfile
        NMAKE /help
        NMAKE [/nologo] [/acdeinpqrst?] [/f makefile] [/x
stderrfile]
        [macrodefs][targets]

Where the options stand for
  /a     force All targets to be built
  /c     Cryptic mode; suppress sign-on banner & warning messages
  /d     Display modification dates
  /e     Environment variables override macros in the makefile
  /i     Ignore exit codes of commands invoked
  /n     No execute mode; display commands only
  /p     Print macro definitions & target descriptions
  /q     Query if target is up to date; for use in batch files
  /r     inference Rules from 'tools.ini' to be ignored
  /s     Silent execution of commands
  /t     Touch targets with current date & time
  /?     Help message
  /help   Help message
  /nologo Do not display sign-on banner
```

Using NMAKE Command Files

A command file is a *response file* used to extend command-line input to NMAKE.

You can split input to NMAKE between the command line and a command file. Use the name of a command file (preceded by @) where you normally type the input information on the command line.

Why Use a Command File?

Use a command file for:

- Complex and long commands you type frequently

- Strings of command-line arguments, such as macro definitions, that exceed the limit for command-line length

  Note: A command file is not the same as a description file. For information about description files, see Description Files

Command-File Syntax

To provide input to NMAKE with a command file, type

```
NMAKE @commandfile
```

For the <commandfile> parameter, enter the name of a file containing the same information as is normally entered on the command line.

NMAKE treats line breaks that occur between arguments as spaces. Macro definitions can span multiple lines if you end each line except the last with a backslash (\). Macro definitions that contain spaces must be enclosed by quotation marks, just as if they were entered directly on the command line.

Example

The following is a command file called UPDATE:

```
/S "program \
= flash" SORT.EXE SEARCH.EXE
```

You can use this command file by typing the following command:

```
NMAKE @UPDATE
```

This runs NMAKE using:

- The /S option
- The macro definition "program = flash"
- The targets specified as SORT.EXE and SEARCH.EXE
- The description file, MAKEFILE, by default

Note that the backslash allows the macro definition to span two lines.

Options

The following describes the options you can use with NMAKE.

Keep the following in mind when using these options:

- Option characters are not case-sensitive; /I and /i are equivalent.
- You can use either a slash or a dash before the option characters; -a and /a are equivalent.

Produce Error File (/X)

Syntax: `/X stderrfile`

This option produces a standard error file.

Build All Targets (/A)

Syntax: /A

This option builds all specified targets, even if they are not out-of-date with respect to their dependent files.

See Description Files.

Suppress Messages (/C)

Syntax: `/C`

This option suppresses display of the NMAKE sign-on banner, non-fatal error messages, and warning messages. To suppress the sign-on banner without suppressing other messages, use the /NOLOGO option.

Display Modification Dates (/D)

Syntax: `/D`

This option displays the modification date of each file when the dates of target and dependent files are checked.

See Description Files.

Override Environment Variables (/E)

Syntax: `/E`

This option disables inherited macro redefinition.

NMAKE *inherits* all current environment variables as macros, which can be redefined in a description file. The /E option disables any redefinition - the inherited macro always has the value of the environment variable.

Specify Description File (/F)

Syntax: `/F filename`

This option specifies <filename> as the name of the description file to use. If a dash (-) is entered instead of a file name, NMAKE reads a description file from the standard input device, typically the keyboard.

If a file name is not specified, it defaults to MAKEFILE.

Display Help (/HELP or /?)

Syntax: `/HELP` OR `/?`

This option displays a brief summary of NMAKE syntax.

Ignore Exit Codes (/I)

Syntax: /I

This option ignores exit codes (also called error level or return codes) returned by programs such as compilers or linkers called by NMAKE. If this option is not specified, NMAKE ends when any program returns a nonzero exit code.

Display Commands (/N)

Syntax: /N

This option causes NMAKE commands to be displayed but not executed. Use the /N option to:

- Check which targets are out-of-date with respect to their dependents
- Debug description files

Suppress Sign-On Banner (/NOLOGO)

Syntax: /NOLOGO

This option suppresses the sign-on banner display when NMAKE is started. If you want to suppress non fatal error messages and warnings as well, use the suppress messages (/C) option.

Print Macro and Target Definitions (/P)

Syntax: /P

This option writes all macro definitions and target definitions. Output is sent to the standard output device (typically the display).

Return Exit Code (/Q)

Syntax: /Q

This option causes NMAKE to return either of the following:

- A 0 exit code if all targets built during an NMAKE run are up-to-date
- A nonzero exit code if they are not up-to-date

Use this option to run NMAKE from within a batch file.

Ignore TOOLS.INI File (/R)

Syntax: /R

This option ignores the following:

- All inference rules and macros contained in the TOOLS.INI file
- All predefined inference rules and macros

Suppress Command Display (/S)

Syntax: /S

This option suppresses the display of commands as they are executed by NMAKE. It does not suppress the display of messages generated by the commands themselves.

The /N command (Display Commands) takes precedence over the /S option. If you use /N and /S together, commands are displayed but not executed.

Change Target Modification Dates (/T)

Syntax: /T

This option changes or "touches" the modification dates for out-of-date target files to the current date. No commands are executed, and the target file is left unchanged.

Description Files

NMAKE uses a description file to determine what to do. In its simplest form, a description file tells NMAKE which files depend on others and which commands need to be executed if a file changes.

A description file looks like this:

```
targets...: dependents...
     command                 description block
        :

targets... : dependents...
     command
```

Description Blocks

A dependent relationship between files is defined in a description block. A "description block" indicates the relationship among various parts of the program. It contains commands to bring all components up to date. The description file can contain up to 1024 description blocks.

```
Description File                      Description Block
```

```
Description
  Block 1                          targets... : dependents...
                                    command
Descr Blk 2                         command
                                    command
     :                                 :

Descr Blk n
```

Special Features

The following are special features of description blocks:

- Description files can contain macro definitions and use macros in description blocks. Macros allow easy substitution of one text string for another.
- Description files can contain inference rules. Inference rules allow NMAKE to infer which commands to execute based on the file-name extensions used for targets and dependents.
- You can specify directories for NMAKE to search for dependent files by using the following syntax:

  ```
  targets : {directory1;directory2...}dependent ...
  ```

  NMAKE searches the current directory first, then *<directory1>*, *<directory2>*, and so on.
- A command can be placed on the same line as the target and dependent files by using a semicolon (;) as depicted below:

  ```
  targets... : dependents... ; command
  ```

- A long command can span several lines if each line ends with a backslash (\ ):

  ```
  command \
    continuation of command
  ```

- The execution of a command can be modified if you precede the command with special characters.
- If you do not specify a command in a description block, NMAKE looks for an inference rule to build the target.
- DOS and OS/2 wildcard characters (* and ?) can be used in description blocks. For example, the following description block compiles all source files with the .C extension:

  ```
  ASTRO.EXE : *.C
    ICC $**
  ```

- NMAKE will expand the **.C* specification into the complete list of C files in the current directory. *$*** is a complete list of dependents specified for the current target.
- NMAKE uses several punctuation characters in its syntax. To use one of these characters as a literal character, place an escape character (^) in front of it. For a list of punctuation characters, see Escape Characters.
- Normally a target file can appear in only one description block. A special syntax allows you to use a target in several description blocks.
- A special syntax allows you to determine the drive, path, base name, and extension of the first dependent file in a description block.

Targets in Several Description Blocks

Using a file as a target in more than one description block causes NMAKE to end. You can overcome this limitation by using two colons (::) as the target/dependent separator instead of one colon.

The following description block is permissible:

```
X :: A
  command
X :: B
  command
```

The following causes NMAKE to end:

```
X : A
  command
X : B
  command
```

It is permissible to use single colons if the target/dependent lines are grouped above the same commands. The following is permissible:

```
X : A
X : B
  command
```

Double Colon (::) Target/Dependent Separator Example

```
TARGET.LIB :: A.ASM B.ASM C.ASM
  ML A.ASM B.ASM C.ASM
  LIB TARGET -+A.OBJ -+B.OBJ -+C.OBJ;

TARGET.LIB :: D.C E.C
  ICC /C D.C E.C
  LIB TARGET -+D.OBJ -+E.OBJ;
```

These two description blocks update the library named TARGET.LIB. If any of the assembly-language files have changed more recently than the library file, NMAKE executes the commands in the first block to assemble the source files and update the library. Similarly, if any of the C-language files have changed, NMAKE executes the second group of commands to compile the C files and update the library.

Macros

Macros provide a convenient way to replace one string with another in the description file. The text is automatically replaced each time NMAKE is run. This feature makes it easy to change text throughout the description file without having to edit every line that uses the text. Two common uses of macros are:

- To create a standard description file for several projects. The macro represents the file names in commands. These file names are defined when you run NMAKE. When you switch to a different project, changing the macro changes the file names NMAKE uses throughout the description file.
- To control the options that NMAKE passes to the compiler, assembler, or linker. When using a macro to specify the options, you can quickly change the options throughout the description file in one easy step.

A macro can be defined:

- In a description file
- On the command line
- In TOOLS.INI
- Through inheritance from environment variables

Example

```
program = FLASH
c = LINK
options =

$(program).EXE : $(program).OBJ
  $c $(options) $(program).OBJ;
```

The example above defines three macros. The description block executes the following commands:

```
FLASH.EXE : FLASH.OBJ
  LINK  FLASH.OBJ;
```

Special Features

Macros have the following special features:

- When using a macro, you can substitute text in the macro itself.
- Several macros have been predefined for special purposes.
- If a macro is defined more than once, precedence rules govern which definition is used.
- You can also put macros into your TOOLS.INI file.

Macros in a Description File

Before using a macro, you need to define it, either on the NMAKE command line or in your description file. Description-file macro definitions look like this:

```
macroname = macrostring
```

Macro names can be any combination of alphanumeric characters and the underscore character (_), and they are case-sensitive. A macro string can be any string of characters.

The first character of the macro name must be the first character on the line. NMAKE ignores any spaces before or after the equal sign (=).

The macro string can be a null string and can contain embedded spaces. Do not enclose the macro string in quotation marks; quotation marks are used only when you define macros on the command line.

Macros on the Command Line

Before using a macro, you need to define it, either on the NMAKE command line or in your description file. Command-line macro definitions look like this:

```
macroname=macrostring
```

No spaces can surround the equal sign. If you embed spaces, NMAKE might misinterpret your macro. If your macro string contains

embedded spaces, enclose it in double quotation marks (") like this:

```
macroname="macro string"
```

or simply enclose the entire macro definition in double quotation marks (") like this:

```
"macroname = macro string"
```

Macro names can be any combination of alphanumeric characters and the underscore character (_), and they are case-sensitive. A macro string can be any string of characters or a null string.

Inherited Macros

NMAKE *inherits* all current environment variables as macros. For example, if you have a PATH environment variable defined as `PATH = C:\TOOLS\BIN`, the string `C:\TOOLS\BIN` is substituted when you use PATH in the description file.

You can redefine inherited macros by including a line such as the example above in a description file. While NMAKE is running, the macro takes on the redefined definition. When NMAKE terminates, however, the environment variable resumes its original value.

The Override Environment Variables (/E) option disables inherited macro redefinition. If you use this option, NMAKE ignores any attempt to redefine an inherited macro.

Defined Macros

After you have defined a macro, you can use it anywhere in your description file with the following syntax:

```
$(macroname)
```

The parentheses are not required if the macro name is only one character long. To use a dollar sign ($) without using a macro, enter two dollar signs ($$), or use the caret (^) before the dollar sign as an escape character.

When NMAKE runs, it replaces all occurrences of *$(macroname)* with the defined macro string. If the macro is undefined, nothing is substituted. After a macro is defined, you can cancel it only with the !UNDEF directive.

Macro Substitutions

Just as you use macros to substitute text within a description file, you use the following syntax to substitute text within a macro:

```
$(macroname: string1 = string2)
```

Every occurrence of *<string1>* is replaced by *<string2>* in *<macroname>*. Spaces between the colon and *<string1>* are considered part of *<string1>*. If *<string2>* is a null string, all occurrences of *<string1>* are deleted from the macro. The colon (:) must immediately follow *<macroname>*.

Note: The replacement of *<string1>* with *<string2>* in the macro is not a permanent change. If you use the macro again without a substitution, you get the original unchanged macro.

Example

```
SOURCES = ONE.C TWO.C THREE.C
PROGRAM.EXE : $(SOURCES:.C=.OBJ)
```

```
  LINK $**;
```

The example above defines a macro called SOURCES, which contains the names of three C source files. With this macro, the target/dependent line substitutes the .OBJ extension for the .C extension. Thus, NMAKE executes the following command:

```
LINK ONE.OBJ TWO.OBJ THREE.OBJ;
```

Note: *$*** is a special macro that translates to all dependent files for a given target.

Special Macros

NMAKE predefines several macros. The first six macros below return one or more file specifications for the files in the target/dependent line of a description block. Except where noted, the file specification includes the path of the file, the base file name, and the file-name extension.

Macro	Value
$@	The specification of the target file.
$*	The base name (without extension) of the target file. Path information is also returned if the path was specified as part of the target file name. This macro cannot be used in a dependent list.
$**	The specifications of the dependent files.
$?	The specifications for only those dependent files that are out-of-date with respect to the targets.
$<	The specification of a single dependent file that is out-of-date with respect to the targets. This macro is used only in inference rules.
$$@	The file specification of the target that NMAKE is currently evaluating. This is a dynamic dependency parameter, used only in dependent lists.
$(CC)	The string ICC, which is the command to run the C Set ++ Compiler. You can redefine this macro to use a different command.
$(AS)	The string MASM, which is the command to run the Macro Assembler (MASM). You can redefine this macro to use a different command.
$(MAKE)	The command name used to run NMAKE. This macro is used to invoke NMAKE recursively. If you redefine this macro, NMAKE issues a warning message. **Note:** NMAKE executes the command line in which $(MAKE) appears, even if the display commands (/N) option is on.
$(MAKEFLAGS)	The NMAKE options currently in effect. You cannot redefine this macro.

Note: The special macros $** and $$@ are the only exceptions to the rule that macro names longer than one character must be enclosed in parentheses.

You can append characters to any of the first six macros in this list to modify the meaning of the macro. However, you cannot use macro substitutions in these macros.

Examples

```
TRIG.LIB : SIN.OBJ COS.OBJ ARCTAN.OBJ
```

```
  !LIB TRIG.LIB -+$?;
```

In the example above, the macro $? represents the names of all dependent files that are out-of-date with respect to the target file. The exclamation point (!) preceding the LIB command causes NMAKE to execute the LIB command once for each dependent file in the list. As a result of this description, the LIB command is executed up to three times, each time replacing a module with a newer version.

```
DIR=C:\INCLUDE
$(DIR)\GLOBALS.H : GLOBALS.H
 COPY GLOBALS.H $@
$(DIR)\TYPES.H : TYPES.H
 COPY TYPES.H $@
$(DIR)\MACROS.H : MACROS.H
 COPY MACROS.H $@
```

The example above shows how to update a group of include files. Each of the files GLOBALS.H, TYPES.H, and MACROS.H in the directory C:\INCLUDE depends on its counterpart in the current directory. If one of the include files is out-of-date, NMAKE replaces it with the file of the same name from the current directory.

The following description file, which uses the special macro $$@, is equivalent:

```
DIR=C:\INCLUDE
$(DIR)\GLOBALS.H $(DIR)\TYPES.H $(DIR)\MACROS.H : $$(@F)
!COPY $? $@
```

The special macro $$(@F) signifies the file name (without the path) of the current target.

When NMAKE evaluates the description block, it evaluates the three targets, one at a time, with respect to their dependents. Thus, NMAKE first checks whether C:\INCLUDE\GLOBALS.H is out-of-date compared with GLOBALS.H in the current directory. If so, it executes the command to copy the dependent file GLOBALS.H to the target. NMAKE repeats the procedure for the other two targets.

Note that on the command line, the macro $? refers to the dependent for this target. The macro $@ specifies the full file specification of the target file.

File-Specification Parts

A full file specification gives the base name of the file, the file-name extension, and the path. The path provides the disk-drive identifier and the sequence of directories needed to locate the file on the disk.

For example, the file specification:

```
C:\SOURCE\PROG\SORT.OBJ
```

has the following parts:

```
Path Name            C:\SOURCE\PROG
Base File Name       SORT
File-Name Extension  .OBJ
```

Characters That Modify Special Macros

The following six macros all resolve to a file specification (or possibly several file specifications for $** and $?):

```
     $*      $@      $**      $<      $?      $$@
```

You can append characters to any of these macros to modify the file name returned by the macro. Depending on which character you use, parts of the full file specification are returned:

	Appended Character			
File Part Returned	D	F	B	R
File Path	Yes	No	No	Yes
Base File Name	No	Yes	Yes	Yes
File Name Extension	No	Yes	No	No

Modified Special Macros Example

If the macro $@ has the value

```
C:\SOURCE\PROG\SORT.OBJ
```

the following values are returned for the modified macro:

Macro	Value
$(@D)	C:\SOURCE\PROG
$(@F)	SORT.OBJ
$(@B)	SORT
$(@R)	C:\SOURCE\PROG\SORT

Note: Modified macros are always longer than a single character - they must be enclosed by parentheses when used.

Macro Precedence Rules

When the same macro is defined in more than one place, the definition with the highest priority is used:

1. (Highest) Command line
2. Description file
3. TOOLS.INI file
4. Environment variables
5. (Lowest) Predefined macros (such as CC and AS)

If you invoke NMAKE with the Overriding Macro Definitions (/E) option, macros defined by environment variables take precedence over those defined in a description file.

Inference Rules

Inference rules are templates from which NMAKE infers what to do with a description block when no commands are given. Only those extensions defined in a .SUFFIXES list can have inference rules. The extensions .C, .OBJ, .ASM, and .EXE are automatically included in .SUFFIXES.

When NMAKE encounters a description block with no commands, it looks for an inference rule that specifies how to create the target from the dependent files, given the two file extensions. Similarly, if a dependent file does not exist, NMAKE looks for an inference rule that specifies how to create the dependent file from another file with the same base name.

NMAKE applies an inference rule only if the base name of the file it is trying to create matches the base name of a file that already exists.

In effect, inference rules are useful only when there is a one-to-one correspondence between the files with the "from" extension and the files with the "to" extension. You cannot, for example, define an inference rule that inserts a number of modules into a library.

The use of inference rules eliminates the need to put the same commands in several description blocks. For example, you can use inference rules to specify a single ICC command that changes any C source file (with a .C extension) to an object file (with a .OBJ extension).

You define an inference rule by including text of the following form in your description file or in your TOOLS.INI file - see "Special Features".

```
.fromext.toext:
commands
:
```

The elements of the inference rule are:

<fromext>
: The file-name extension for dependent files to build a target

<toext>
: The file-name extension for target files to be built

<commands>
: The commands to build the <toext> target from the <fromext> dependent.

For example, an inference rule to convert C source files (with the .C extension) to C object files (with the .OBJ extension) is

```
.C.OBJ:
 ICC $<
```

Note: The special macro $< represents the name of a dependent file that is out-of-date relative to the target.

Special Features

- You can specify a path where NMAKE should look for target and dependent files used in inference rules.
- Inference rules are predefined for compiling and linking C programs, and for assembling programs.
- NMAKE looks for inference rules in the TOOLS.INI file if it cannot find a rule in a description file.
- Only those extensions defined in a .SUFFIXES list can have inference rules. The extensions .C, .OBJ, .ASM, and .EXE are automatically included in .SUFFIXES.

Example

```
.OBJ.EXE:
  LINK $<;

EXAMPLE1.EXE: EXAMPLE1.OBJ

EXAMPLE2.EXE: EXAMPLE2.OBJ
  LINK /CO EXAMPLE2,,,LIBV3.LIB
```

The first line above defines an inference rule that causes the LINK command to create an executable file whenever a change is made in the

corresponding object file. The file name in the inference rule is specified with the special macro $< so that the rule applies to any .OBJ file with an out-of-date executable file.

When NMAKE does not find any commands in the first description block, it checks for a rule that might apply and finds the rule defined on the first two lines of the description file. NMAKE applies the rule, replacing $< with EXAMPLE1.OBJ when it executes the command, so that the LINK command becomes

```
LINK EXAMPLE1.OBJ;
```

NMAKE does not search for an inference rule when examining the second description block, because a command is explicitly given.

Inference-Rule Path Specifications

When defining an inference rule, you can indicate to NMAKE where to look for target and dependent files. Use the following syntax:

```
{frompath}.fromext{topath}.toext
 commands
 :
```

NMAKE looks in the directory specified by *<frompath>* for files with the *<fromext>* extension. It executes the commands to build files with the *<toext>* extension in the directory specified by *<topath>*.

Predefined Inference Rules

NMAKE predefines three inference rules:

Inference Rule	Default	Command Action
.C.OBJ	$(CC) $(CFLAGS) /C $*.C	ICC /C $*.C
.C.EXE	$(CC) $(CFLAGS) $*.C	ICC $*.C
.ASM.OBJ	$(AS) $(AFLAGS) $*;	MASM $*;

NOTE

1. The first two rules automatically compile and link C programs.
2. The last rule automatically assembles programs.

Directives

Using directives, you can construct description files similar to batch files. NMAKE provides directives that:

- Conditionally execute commands
- Display error messages
- Include the contents of other files
- Turn some NMAKE options on or off

Each directive begins with an exclamation point (!) in the first column of the description file. Spaces can be placed between the exclamation point and the directive keyword.

The list below describes the directives:

!IF <expression>

Executes the statements between the !IF keyword and the next !ELSE or !ENDIF directive if *<expression>* evaluates to a nonzero value.

The *<expression>* used with the !IF directive can consist of integer constants, string constants, or exit codes returned by programs. Integer constants can use the C unary operators for numerical negation (-), one's complement (~), and logical negation (!). You can also use any of the C binary operators listed below:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
&	Bitwise AND
|	Bitwise OR
^^	Bitwise XOR
&&	Logical AND
||	Logical OR
<<	Left shift
>>	Right shift
==	Equality
!=	Inequality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Note:

You can use parentheses to group expressions.

Values are assumed to be decimal values unless specified with a leading 0 (octal) or leading 0x (hexadecimal).

Strings are enclosed by quotation marks ("). You can use the equality (==) and inequality (!=) operators to compare two strings.

You can invoke a program in an expression by enclosing the program name in square brackets ([ ]). The exit code returned by the program is used in the expression.

!ELSE

Executes the statements between the !ELSE and !ENDIF directives if the statements preceding the !ELSE directive were not executed.

!ENDIF

Marks the end of the !IF, !IFDEF, or !IFNDEF block of statements.

!IFDEF <macroname>

Executes the statements between the !IFDEF keyword and the next !ELSE or !ENDIF directive if *<macroname>* is defined in the description file. If a macro has been defined as null, it is still considered to be defined.

!IFNDEF <macroname>

Executes the statements between the !IFNDEF keyword and the next !ELSE or !ENDIF directive if *<macroname>* is not defined

in the description file.

!UNDEF <macroname>
Undefines a previously defined macro.

!ERROR <text>
Prints text and then stops execution.

!INCLUDE <filename>
Reads and evaluates the file *<filename>* before continuing with the current description file. If *<filename>* is enclosed by angle brackets (<>), NMAKE searches for the file in the directories specified by the INCLUDE macro; otherwise, it looks only in the current directory. The INCLUDE macro is initially set to the value of the INCLUDE environment variable.

!CMDSWITCHES {+|-}<opt>
Turns on or off one of four NMAKE options: /D, /I, /N, and /S. If no options are specified, the options are reset to the values they had when NMAKE was started. To turn an option on, precede it with a plus sign (+); to turn it off, precede it with a minus sign (-). This directive updates the MAKEFLAGS macro.

See Special Macros.

Example

```
!INCLUDE <INFRULES.TXT>
!CMDSWITCHES +D
WINNER.EXE:WINNER.OBJ
!IFDEF DEBUG
! IF "$(DEBUG)"=="y"
     LINK /CO WINNER.OBJ;
! ELSE
     LINK WINNER.OBJ;
! ENDIF
!ELSE
! ERROR Macro named DEBUG is not defined.
!ENDIF
```

The directives in this example do the following:

- The !INCLUDE directive causes the file INFRULES.TXT to be read and evaluated as if it were part of the description file.
- The !CMDSWITCHES directive turns on the /D option, which displays the dates of the files as they are checked.
- If WINNER.EXE is out-of-date with respect to WINNER.OBJ, the !IFDEF directive checks to see whether the macro DEBUG is defined. If it is defined, the !IF directive checks to see whether it is set to y. If it is, the linker is invoked with the /CO option; otherwise, it is invoked without the /CO. If the DEBUG macro is not defined, the !ERROR directive prints the message and NMAKE stops executing.

Pseudotargets

A pseudotarget is a target in a description block that is not a file. Instead, it is a name that serves as a *handle* for building a group of files or executing a group of commands. In the following example, UPDATE is a pseudotarget:

```
UPDATE: *.*
  !copy $** A:\PRODUCT
```

When NMAKE evaluates a pseudotarget, it always considers the dependent files to be out-of-date. In the description above, NMAKE copies each of the dependent files to the specified drive and directory.

NMAKE predefines several pseudotargets for special purposes.

See Predefined Pseudotargets.

Predefined Pseudotargets

NMAKE predefines several pseudotargets that provide special rules within a description file:

.SILENT

Syntax: .SILENT : dependents...

This pseudotarget suppresses the display of executed commands for a single description block. The /S option does the same thing for all description blocks.

See Suppress Command Display (/S).

.IGNORE

Syntax: .IGNORE : dependents...

This pseudotarget ignores exit codes returned by programs for a single description block. The /I option does the same thing for all description blocks.

See Ignore Exit Codes (/I).

.SUFFIXES

Syntax: .SUFFIXES : extensions...

This pseudotarget defines file extensions to try when NMAKE needs to build a target file for which no dependent files are specified. NMAKE searches the current directory for a file with the same name as the target file and an extension in *<extensions...>*. If NMAKE finds such a file, and if an inference rule applies to the file, NMAKE treats the file as a dependent of the target.

The .SUFFIXES pseudotarget is predefined as

```
.SUFFIXES : .OBJ .EXE .C .ASM
```

To add extensions to the list, specify .SUFFIXES : followed by the new extensions. To clear the list, specify

```
.SUFFIXES:
```

Note: Only those extensions specified in .SUFFIXES can have inference rules. NMAKE ignores inference rules unless the extensions have been specified in a .SUFFIXES list.

.PRECIOUS

Syntax: .PRECIOUS : targets...

This pseudotarget tells NMAKE not to delete a target even if the commands that build it are terminated or interrupted. This pseudotarget overrides the NMAKE default. By default, NMAKE deletes the target if it cannot be sure that the target was built successfully.

For example,

```
.PRECIOUS : TOOLS.LIB
TOOLS.LIB : A2Z.OBJ Z2A.OBJ
 command
  :
```

If the commands to build TOOLS.LIB are interrupted, leaving an incomplete file, NMAKE does not delete the partially built TOOLS.LIB.

Note: The pseudotarget .PRECIOUS is useful only in limited circumstances. Most professional development tools have their own interrupt handlers and "clean up" when errors occur.

InLine Files

You might need to issue a command in the description file with a list of arguments exceeding the command-line limit of the operating system. Just as NMAKE supports the use of command files, it can also generate inline files that are read as response files by other programs.

To generate an inline file, use the following syntax for your description block:

```
target : dependents
  command @<<[filename]
inline file text
<< [KEEP | NOKEEP]
```

All of the text between the two sets of double less-than signs (<<) is placed into an inline file and given the name *<filename>.* You can refer to the inline file at a later time by using *<filename>*. If *<filename>* is not given, NMAKE gives the file a unique name in the directory specified by the TMP environment variable, if it is defined. Otherwise, NMAKE creates a unique file name in the current directory.

The inline file can be temporary or permanent. If you do not specify otherwise, or if you specify the keyword NOKEEP, the inline file is temporary. Specify KEEP to retain the file.

Note: The at sign (@) is not part of the NMAKE syntax but is the typical character used by utility programs (such as LINK386) to designate a file as a response file.

Example

```
MATH.LIB :  ADD.OBJ SUB.OBJ MUL.OBJ DIV.OBJ
  LIB @<<
MATH.LIB
-+ADD.OBJ-+SUB.OBJ-+MUL.OBJ-+DIV.OBJ
listing
<<
```

The above example creates an inline file and uses it to invoke the Library Manager (LIB). The inline file is used as a response file by (LIB). It specifies which library to use, the commands to execute, and the listing file to produce. The inline file contains:

```
MATH.LIB
-+ADD.OBJ-+SUB.OBJ-+MUL.OBJ-+DIV.OBJ
listing
```

Because no file name is listed after the LIB command, the inline file is given a unique name and placed into the current directory (or the directory defined by the TMP environment variable).

Escape Characters

NMAKE uses the following punctuation characters in its syntax:

```
(       )       #       $       ^       \
{       }       !       @       -
```

To use one of these characters in a command and not have it interpreted by NMAKE, use a caret (^) in front of the character.

For example,

```
BIG^#.C
```

is treated as

```
BIG#.C
```

With the caret, you can include a literal newline character in a description file. This capability is useful in macro definitions, as in the following example:

```
XYZ=abc^<ENTER>
def
```

The effect is equivalent to assigning the C-style string "abc\ndef" to the XYZ macro. Note that this effect differs from the effect of using the backslash (\ ) to continue a line. A newline character that follows a backslash is replaced with a space.

NMAKE ignores a caret that is not followed by any of the characters it uses in its syntax. A caret that appears within quotation marks is not treated as an escape character.

Note: The escape character cannot be used in the command portion of a dependency block.

Characters That Modify Commands

Any of three characters can be placed in front of a command to modify how the command is run:

- (dash)	Turns off error checking for the command
@ (at sign)	Suppresses display of the command
! (exclamation point)	Executes the command for each dependent file

Note:

Spaces can separate the modifying character from the command. Any command on a separate line - whether modified or not -

must be indented by one or more spaces or tabs.

You can use more than one character to modify a single command.

Turn Error Checking Off (-)

Syntax: -[n] command

The /I option globally turns command error-checking off. The dash (-) command modifier overrides the global setting to turn error checking off for commands individually. This modifier is used in two ways:

- A dash without a number turns off all error checking.
- A dash followed by a number causes NMAKE to halt only if the exit code returned by the command is greater than the number.

See Ignore Exit Codes (/I).

Dash-Command Modifier Examples

```
LIGHT.LST : LIGHT.TXT
  - FLASH LIGHT.TXT
```

In the example above, NMAKE never ends, regardless of the exit code returned by FLASH.

```
LIGHT.LST : LIGHT.TXT
  -1 FLASH LIGHT.TXT
```

In the example above, NMAKE ends if the exit code returned by FLASH is greater than 1.

Suppress Command Display (@)

Syntax: @ command

The /S option globally suppresses the display of commands while NMAKE is running. The at sign (@) modifier suppresses the display for individual commands.

Note: Regardless of the /S option or the @ modifier, output generated by the command itself always appears.

See Suppress Command Display (/S).

At Sign (@) Command Modifier Example

Suppress Command Display (@)

```
SORT.EXE:SORT.OBJ
```

```
  @ ECHO sorting
```

The command line calling the ECHO command is not displayed. The output of the ECHO command, however, is displayed.

Execute Command for Dependent Files (!)

Syntax: ! command

The exclamation-point command modifier causes the command to be executed for each dependent file if the command uses one of the special macros $? or $**. The $? macro refers to all dependent files out-of-date with respect to the target. The $** macro refers to all dependent files in the description block.

See Special Macros.

Exclamation Point (!) Command Modifier Examples

```
LEAP.TXT : HOP.ASM SKIP.BAS JUMP.C
  ! print $** lpt1:
```

The example above executes the following three commands, regardless of the modification dates of the dependent file:

```
print HOP.ASM lpt1:
print SKIP.BAS lpt1:
print JUMP.C lpt1:
```

```
LEAP.TXT : HOP.ASM SKIP.BAS JUMP.C
  ! print $? lpt1:
```

The example above executes the print command only for those dependent files with modification dates later than that of the LEAP.TXT file. If HOP.ASM and JUMP.C have modification dates later than LEAP.TXT, the following two commands are executed:

```
print HOP.ASM lpt1:
print JUMP.C lpt1:
```

EXTMAKE Syntax

Description files can use a special syntax to determine the drive, path, base name, and extension of the first dependent file in a description block. This syntax is called the "extmake" syntax.

The characters, %S, represent the complete file specification of the first dependent file. Various parts of the file specification are represented using the syntax

```
%|partsF
```

where *<parts>* is a combination of the following letters:

d	Drive
p	Path
f	Base name

e Extension

For example, to specify the drive and path name of the first dependent file in a description block, use:

%|dpF

The percent symbol (%) is a replacement in DOS and OS/2 command lines. To use the percent symbol in command-line arguments, use a double percent (%%).

Macros and Inference Rules in TOOLS.INI

You can place either macros or inference rules in your TOOLS.INI file. NMAKE looks for the TOOLS.INI file first in the current directory and then in the directory indicated by the INIT environment variable.

If NMAKE finds a TOOLS.INI file, it looks for the following tag:

```
[nmake]
```

You can place macros and inference rules below this tag in the same format you would use in a description file.

If a macro or inference rule is defined in both the TOOLS.INI file and the description file, the definition in the description file takes precedence. Also, if you use the /R option, the TOOLS.INI file is ignored.

Example

```
[nmake]
CFLAGS=/ss /ms /Gd-
.C.OBJ:
  $(CC) -c $(CFLAGS) $*.C
```

These lines in the TOOLS.INI file do the following:

- Define the CFLAGS macro as "/ss /ms /Gd-"
- Redefine the predefined inference rule to build .OBJ files from .C source files

---

NMAKE Error Messages

This section provides a convenient reference to the many error messages that can be encountered when using the NMAKE facility.

Error Message Descriptions

Fatal Error Messages (Part 1) 1000 - 1098

U1000 syntax error : ')' missing in macro invocation
Explanation: A left parenthesis appeared without a matching right parenthesis in a macro invocation. The correct form is $(*name*).
Action: Add the right parenthesis in the proper syntax.

U1001 syntax error : illegal character *'character'* in macro
Explanation: A non-alphanumeric character other than underscore appeared in a macro.
Action: Use only characters valid for a macro name.

U1002 syntax error : bad macro invocation '$'
Explanation: A single dollar sign ($) appeared without a macro name associated with it. The correct form is $(*name*).
Action: Use a defined macro name.

U1003 syntax error : '=' missing in macro
Explanation: The = sign was missing in a macro definition. The correct form is *'name = value'*.
Action: Insert an equals sign (=) and retry.

U1004 syntax error : macro name missing
Explanation: A macro invocation appeared without a name. The correct form is $(*name*).
Action: Supply the macro name and retry.

U1005 syntax error : text must follow ':' in macro
Explanation: A string substitution was specified for a macro, but the string to be changed in the macro was not specified.
Action: Specify the string to be substituted.

U1006 syntax error : missing closing double quotation mark
Explanation: An opening double quotation mark appeared without a closing quotation mark.
Action: Edit the line and add the closing quotation mark.

U1007 double quotation mark not allowed in name
Explanation: You used a ' " ' symbol inside a macro name.
Action: Correct the name without the quotation mark.

U1017 unknown directive *'directive'*
Explanation: The *directive* specified is not a recognized directive.
Action: Check your spelling of the intended directive.

U1018 directive and/or expression part missing
Explanation: The directive is incompletely specified. The expression part is required.
Action: Supply the expression and retry the directive.

U1019 too many nested if blocks
Explanation: You exceeded the limit of 16 levels of nested !IF directives.
Action: Simplify your nesting logic to fewer than 16 levels.

U1020 EOF found before next directive
Explanation: A directive, such as !ENDIF, was missing.
Action: Insert the required directive and retry.

U1021 syntax error : else unexpected
Explanation: An !ELSE directive was found that was not expected, or was placed in a syntactically incorrect place.
Action: Correct the position of the !ELSE directive.

U1022 missing terminating character for string/program invocation : *'character'*
Explanation: The closing double quotation mark in a string comparison in an !IF directive was missing. Or else the closing bracket ( ]) in a program invocation in a directive is missing.
Action: Insert the proper termination character.

U1023 syntax error present in expression
Explanation: An expression is incorrect.
Action: Check the allowed operators and operator precedence for the expression.

U1024 illegal argument to !CMDSWITCHES
Explanation: An unrecognized !CMDSWITCHES option was specified.
Action: Use the correct !CMDSWITCHES option.

U1031 file name missing (or macro is null)
Explanation: An !INCLUDE directive was found, but the name of the file to include is missing.

Action: Supply the name of the file to be included.

U1033 syntax error : *'string'* unexpected
Explanation: The specified *string* is not part of the valid syntax for a makefile.
Action: Correct the line according to the proper syntax.

U1034 syntax error : separator missing
Explanation: The colon that separates targets from dependents is missing.
Action: Insert a colon after the target list.

U1035 syntax error : expected ':' or '=' separator
Explanation: Either a colon, implying a dependency line, or an = sign, implying a macro definition, was expected. This message will be displayed if NMAKE encounters a premature end-of-file character while expecting one of these separators.
Action: Insert the proper separator in the line.

U1036 syntax error : too many names to left of '='
Explanation: Only one string is allowed to the left of a macro definition.
Action: Remove the incorrect text before the = sign.

U1037 syntax error : target name missing
Explanation: A colon (:) was found before a target name was found. At least one target is required.
Action: Insert the correct target name before the colon.

U1038 internal error : lexer
Explanation: The lexer encountered an unexpected condition.
Action: Note the circumstances of the failure and contact IBM Support.

U1039 internal error : parser
Explanation: The parser encountered an unexpected condition.
Action: Note the circumstances of the failure and contact IBM Support.

U1040 internal error : macro expansion
Explanation: An unexpected condition was found during macro expansion.
Action: Note the circumstances of the failure and contact IBM Support.

U1041 internal error : target building
Explanation: An unexpected condition was found during target building.
Action: Note the circumstances of the failure and contact IBM Support.

U1042 internal error : expression stack overflow
Explanation: An expression was too complex to decode using internal memory space.
Action: Note the circumstances of the failure and contact IBM Support.

U1043 internal error : temp file limit exceeded
Explanation: NMAKE required too many temporary files.
Action: Note the circumstances of the failure and contact IBM Support.

U1044 internal error : too many levels of recursion building a target
Explanation: Recursive invocations of NMAKE exceeded available memory.
Action: Note the circumstances of the failure and contact IBM Support.

U1045 *messagetext*
Explanation: NMAKE encountered an unexpected condition.
Action: Note the text of the message and contact IBM Support.

U1046 internal error : out of search handles
Explanation: NMAKE exceeded an internal limit on handles.
Action: Note the circumstances of the failure and contact IBM Support.

U1049 macro too long (max allowed size : 64K)
Explanation: One of your macros expanded to longer than 65 535 bytes.
Action: Recode the macro definition so that it is less than 64K.

U1050 *user-specified text*
Explanation: The message specified with the !ERROR directive is displayed.
Action: Action depends on the defined error condition.

U1051 out of memory
Explanation: NMAKE ran out of space in the far heap.
Action: Note the circumstances of the failure and contact IBM Support.

U1052 file *'filename'* not found

Explanation: The file was not found.
Action: Specify the *filename* properly in the makefile.

U1053 file *'filename'* unreadable
Explanation: The *filename* cannot be read.
Action: Be sure that the file has the appropriate attributes to be read.

U1054 cannot create in-line file *'filename'*
Explanation: The program was unable to generate the specified in-line file as a uniquely-named temporary file.
Action: Be sure your file system has enough space for temporary files.

U1055 out of environment space
Explanation: The environment space limit was reached.
Action: Restart NMAKE with a larger environment space.

U1056 cannot find command processor
Explanation: The command processor CMD.EXE could not be found.
Action: Be sure that the COMSPEC environment variable points to a command processor on your file system.

U1057 cannot delete temporary file *'filename'*
Explanation: The program was unable to delete the specified file.
Action: The file needs to exist and have the write attribute.

U1058 terminated by user
Explanation: You pressed Ctrl+Break to stop NMAKE.
Action: None, the program has stopped.

U1060 unable to close file : *'filename'*
Explanation: NMAKE was unable to close *filename* .
Action: Look for the named file with write attribute on your file system.

U1061 /F option requires a file name
Explanation: You coded command-line option /f but failed to follow it with the name of a description file.
Action: Specify the description file name after the option.

U1062 missing file name with /X option
Explanation: You coded command-line option /x but failed to follow it with the name of a file to receive redirected stderr output.
Action: Give an output error file name after the option.

U1063 missing macro name before '='
Explanation: You coded '=' in a command line macro definition, but failed to supply the name of the macro.
Action: Give the macro name in the definition.

U1064 MAKEFILE not found and no target specified
Explanation: You invoked NMAKE without a /f option, and no file named MAKEFILE was present.
Action: Either create a file named MAKEFILE, or use the /f switch.

U1065 incorrect option *'option'*
Explanation: NMAKE does not use the option which you specified.
Action: Use a valid command line option.

U1070 cycle in macro definition *'macroname'*
Explanation: A cycle was detected in the macro definition specified.
Action: Rewrite the macro to avoid the circular definition.

U1071 cycle in dependency tree for target *'targetname'*
Explanation: A cycle was detected in the target dependency tree.
Action: Check the dependency lists descending from the given target and remove the circular dependency.

U1072 cycle in include files : *'filenames'*
Explanation: A cycle was detected in the tree of included files.
Action: Check the file names included by the given include file and remove the circular inclusion.

U1073 don't know how to make *'filename'*
Explanation: The specified target does not exist and there are no commands to execute or inference rules given for it. Hence NMAKE cannot build it.
Action: Correct the specification of the file, which should exist on your file system.

U1076 name too long
Explanation: The macro name, target name, or build command name would overflow an internal buffer.
Action: Reduce the length of the specified name.

U1077 *'program'* : return code *'value'*
Explanation: The invocation of NMAKE failed with a nonzero return value.
Action: Determine the cause of failure of the specified *program* .

U1078 constant overflow at *'directive'*
Explanation: A constant in *'directive'* expression was too big.
Action: Reduce the size of the specified constant to a *value* within the range of a signed long integer, -2147483648 <= *value* <= 2147483647.

U1079 illegal expression : divide by zero present
Explanation: An expression contains a division by zero.
Action: Remove the undefined division by zero from the expression.

U1080 operator and/or operand out of place : usage illegal
Explanation: The expression uses an operand or operator incorrectly.
Action: Check the allowed set of operators and their precedence.

U1081 *'program'* : program not found
Explanation: NMAKE could not find the external command or program.
Action: Be sure that the *program* is located in the PATH.

U1082 *'command'* : cannot execute command: out of memory
Explanation: NMAKE ran out of memory while running *command* .
Action: Make more memory available while running NMAKE.

U1083 target macro *'macroname'* expands to nothing
Explanation: The expansion of the given macro is a null string.
Action: Correct the definition of the macro.

U1084 cannot create temporary file *'filename'*
Explanation: NMAKE was unable to open the specified temporary file.
Action: Check the filename specification for validity.

U1085 cannot mix implicit and explicit rules
Explanation: A regular target was specified along with the target for a rule (which has the form *.sufx1.sufx2*).
Action: Separate targets built by implicit and explicit inference rules into different lists.

U1086 inference rule cannot have dependents
Explanation: Dependents are not allowed in the definition of an inference rule.
Action: Remove the dependents list from the rule.

U1087 cannot have : and :: dependents for same target
Explanation: A target cannot have both a single-colon and double-colon dependency.
Action: Choose either single-colon or double-colon separator for the target.

U1088 invalid separator on inference rule : '::'
Explanation: Inference rules can use only a single-colon separator.
Action: Use a single-colon dependency for the target.

U1089 cannot have build commands for pseudotarget *'targetname'*
Explanation: Pseudotargets (for example, .PRECIOUS, .SUFFIXES) cannot have build commands specified.
Action: Remove the build commands from the specification of *targetname* .

U1090 cannot have dependents for pseudotarget *'targetname'*
Explanation: The specified pseudotarget, for example, .SILENT, .IGNORE) cannot have a dependent.
Action: Remove the dependent from the specification of *targetname* .

U1092 too many names in rule
Explanation: The rules cannot have more than one pair of extensions (*ext1.ext2*) as a target for the rule.
Action: Use only one pair of extensions in any inference rule.

U1093 cannot mix special pseudotargets
Explanation: It is illegal to list two or more pseudotargets together.
Action: Use only one pseudotarget in any list.

U1094 syntax error : only [no]keep allowed here
Explanation: In a context where only KEEP or NOKEEP is accepted to indicate the desired disposition of the inline file, you used an incorrect string.
Action: Use the correct syntax for In-Line Files.

U1095 expanded command line *'string'* too long
Explanation: After macro expansion, the command line length exceeds 1024 bytes.
Action: Rewrite the command line to stay within a 1024-byte limit.

U1097 extmake syntax usage error, no dependent
Explanation: You used the extmake file syntax on a description block which had no dependent files.
Action: Specify one or more dependent files for the block.

U1098 extmake syntax in *'string'* incorrect
Explanation: The given string contains an extmake syntax error.
Action: Correct the string according to the proper syntax.

Warnings (Part 2) 4001 - 4008

U4001 command file can be invoked only from command line
Explanation: You used an @ symbol on an argument in a command file. You cannot invoke another command file from within a command file.
Action: If the you want '@' to be part of an argument in a command file, you must enclose that argument in quotation marks.

U4002 no match found for wild card *'string'*
Explanation: NMAKE expanded wildcards in the given *string* , but found no files matching the specification.
Action: Check the existence of desired files on your file system.

U4004 too many rules for target *'targetname'*
Explanation: You specified too many inference rules for the specified *targetname* .
Action: Revise your rules specification for *targetname* .

U4005 ignoring rule *'string'* (extension not in .SUFFIXES)
Explanation: You specified an inference rule with a suffix which was not in the .SUFFIXES list.
Action: To use the suffix, be sure to include it in the .SUFFIXES list.

U4006 special macro undefined : *'macroname'*
Explanation: You specified the undefined macro *macroname* .
Action: NMAKE will ignore the undefined *macroname* . You may use only predefined special macros.

U4007 file name *'filename'* too long; truncating to 8.3
Explanation: The specified *filename* is too long for a FAT file system name.
Action: NMAKE will shorten the filename to at most an eight-character name and 3-character extension.

U4008 removed target *'filename'*
Explanation: While deleting non-precious files, NMAKE erased the specified *filename* which was a target.
Action: Check your lists of targets and dependent files to be sure that *filename* is not needed.

Informational Messages (Part 3) 2-6

Message 2 *'filename'* is up-to-date.
Explanation: The specified target *filename* is no older than any of its dependent files.
Action: NMAKE does not need to rebuild this target.

Message 3 ** *'file1'* newer than *'file2'*
Explanation: While reporting creation times of files, NMAKE notes that *file1* was created after *file2* .
Action: [none]

Message 5 touch *'filename'*
Explanation: You specified option /t to touch targets with the current date and time.
Action: NMAKE has updated the creation time of *filename* .

Message 6 *'filename'* target does not exist
Explanation: The specified *filename* could not be found.
Action: NMAKE will rebuild the target.

Object Utility/2 Description

Object Utility/2 is a Workplace-Shell object that provides a facility for registering Workplace Shell classes, creating instances of Workplace Shell classes, and modifying instances of Workplace Shell classes.

The following attributes can be set or modified for instances of Workplace Shell objects:

- Template
- Copy
- Delete
- Rename
- Print
- Link
- Move
- Drag

The attributes modify the behavior of the objects to allow or not allow these actions. For example, the Template attribute allows you to create a template. Some objects do not allow specific behaviors even if the attribute is selected.

A Workplace Shell class must be registered with the Workplace Shell before it will be recognized by Object Utility/2. After the object class is registered, an instance of that class can be created. Object Utility/2 automates these procedures of object-class registration and instantiation. This tool can create an instance of an object from a class that has already been instantiated, or can modify an existing instance.

Registration of a class is performed by opening the main view of Object Utility/2. The class name and DLL name must be provided. The class is not registered if it has been registered previously.

To modify an existing instance, drag the icon representing the class and drop it of Object Utility/2. You can enter the object ID and class name after opening the main view.

After the item to be installed is dropped, a dialog box is displayed for registration and instantiation information.

To destroy an object created by this tool, drag the object and drop it onto the Shredder object on the Workplace Shell Desktop (if the **no drag** and **no delete** options are not selected and the object allows deletion). A mechanism to deregister an object class is not provided with this tool.

Class Name Field

Class name is a list of all the registered classes that have DLLs available on your system. The OS/2 operating system allows classes to be registered without the DLLs available, but Object Utility/2 does not. You can select a class from the list or enter one manually. The **Class Name** field is required when registering a new class, modifying an existing instance that was not dropped on Object Utility/2, or creating a new instance.

DLL Name Field

The DLL name must be a fully qualified path and file name if the DLL is not located in a DLL search path. This field is required if you are registering a class.

Object ID Field

The Object ID must be enclosed in angle brackets (<>). This field is required when you modify an existing object that was not dropped on

Object Utility/2. If you try to create an instance that is not a template, without an Object ID, you will receive a warning. You can create a new instance without an object ID. The Object ID must be unique, if specified, when creating an instance. Templates cannot have an object ID. Instances with an object ID cannot be made into a template.

Title Field

The Title field is required when creating a new object. You can alter the title of an existing object by providing a different title in this field.

Location Field

You can either select an existing location from the location list or enter a location manually. The location must be an object ID that represents a folder (enclosed in angle brackets) or a fully qualified path name.

Options

Create Instance	Creates an instance of the class.
Template	Creates a template of the class in the Templates folder.
No Copy	Removes **Copy** from the pop-up menu.
No Delete	Removes **Delete** from the pop-up menu.
No Rename	Removes **Rename** from the pop-up menu.
No Print	Removes **Print** from the pop-up menu.
No Link	Removes **Link** from the pop-up menu.
No Move	Removes **Move** from the pop-up menu.
No Drag	Prevents dragging of the object.

PACK/UNPACK and PACK2/UNPACK2

PACK reduces the size of a file by compressing its data. You can use PACK for a single file or for a group of files, thereby reducing the disk space required for your OS/2 application. UNPACK restores a packed file to it's original size.

In addition to the original versions of these utility programs, enhanced versions of the programs are also included with OS/2. These enhanced versions are named PACK2 and UNPACK2. The options, parameters, and function for PACK2 and UNPACK2 are identical to PACK and UNPACK. The only difference is that PACK2 has a better compression algorithm.

Note: PACK and PACK2 are shipped with the OS/2 Toolkit. UNPACK and UNPACK2 are shipped with OS/2.

Starting PACK

You start PACK with a single command from the command line. The input required can be specified in one of two ways:

- You can type the names of all the files you want to compress (Method 1). See Running PACK With Individual Files.
- You can type the name of a single file that contains a list of all the files you want to compress (Method 2). See Running PACK With a Listfile.

When using PACK, select the method that is suitable for you.

Running PACK With Individual Files

You can start PACK with a single command from the command line. You can type the names of all the files you want to compress directly on the command line. Include the drive and path if the files are not in the working directory. You can specify file names with any combination of uppercase and lowercase letters. File-name extensions are not required; however, if you specify a file name that has an extension, also type the extension.

The command-syntax is as follows:

```
PACK sourcefile [packedfile]
  [/H:headerpath\
  |/H:headerfile
  |/H:headerpath\ headerfile]
  [/D:headerdate]
  [/T:headertime]
  [/C] [/A] [/R]
```

where:

sourcefile
: Specifies the name of the file you want packed (compressed). This parameter is required. Include the drive and path if the file is not in the working directory. Global file-name characters are permitted.

  When the data is compressed, the name of the source file is placed in the header of the compressed file and is used as the destination file name during unpacking.

packedfile
: Specifies the name of the file that will contain the compressed data. Files that contain compressed data can be recognized by the @ symbol as the last character in the file name. If you do not specify this parameter, PACK places the compressed data in *sourcefile* and modifies its name to contain the @ symbol.

/H:headerpath\ or /H:headerfile or /H:headerpath\headerfile
: These parameters can be used separately or they can be paired.

  /H:headerpath\
  : Specifies the destination path to be placed in the header of the file that contains the compressed data. Drive letters are not permitted. Unless this path is overridden with the UNPACK command, it will be the destination path when the file is uncompressed. *headerpath* must end with a back slash (\).

  /H:headerfile
  : Specifies the name of the file to be placed in the header of the compressed file. This file name will be used as the destination file for the uncompressed data and cannot be overridden.

    If a header file name is not specified, PACK automatically uses *sourcefile* as the name of the file that is placed in the header of the compressed file.

  /H:headerpath\headerfile
  : Specifies that both a destination path and a destination file name are to be placed in the header of the file that has the compressed data.

/D:headerdate
: Records the date in the header of the file that has the compressed data, and also in the destination file when it is uncompressed.

	The date must follow the format /D:MM-DD-YYYY (for example: /D:08-20-1991 and /D:12-30-2010)
/T:headertime	Records the time in the header of the file that has the compressed data, and also in the destination file when it is uncompressed. The time must follow the format /T:HH.MM (for example /T:02.06 and /T:14.54). Hour 00 represents 12 a.m. and hour 12 represents 12 p.m.
/C	Specifies that the current path be placed in the header of the file that contains the compressed data. When the UNPACK command is used, this path will be the destination path for the file that contains the uncompressed data. You cannot use /C when the *headerpath* is used.
/A	Adds data from *sourcefile* to the data in *packedfile* . The source file can be either in a compressed or uncompressed state. If the source file is in an uncompressed state, the data is compressed before being added to the file containing the compressed data.
/R	Removes the file specified by *sourcefile* from the file that contains only compressed data. The *sourcefile* parameter must specify the path and file name exactly as they appear in the header of the file with the compressed data; otherwise, the following error message appears on the screen. `The specified file to remove was not found.` The /R parameter is valid only when used in conjunction with *sourcefile* and *packedfile* .

Note: To display the path and file-name information stored in the header of the file that contains the compressed data, use the UNPACK command and specify the SHOW option. For information about the SHOW option, see the UNPACK command in the online *OS/2 Command Reference* .

You can also get end-to-end compressed data by using global file-name characters. For example:

```
PACK *.EXE BUNDLE
```

Running PACK With a Listfile

You can start PACK with a single command from the command line. You can type the name of a single file that contains a list of all the files you want to compress.

Include the drive and path if the files are not in the working directory. You can specify file names with any combination of uppercase and lowercase letters. File-name extensions are not required; however, if you specify a file name that has an extension, also type the extension.

The command-line syntax is as follows:

```
PACK listfile [packedfile] /L
  [/H:headerpath\
  |/H:headerfile
  |/H:headerpath\ headerfile]
  [/D:headerdate]
  [/T:headertime]
  [/C]
```

where:

listfile	Specifies the name of the file that contains a list of files that are to be compressed. When naming a list file, do not use global file-name characters. For information about list files, see Creating a List File.

packedfile

Specifies the name of the file that will contain the compressed data. Files that contain compressed data can be recognized by the @ symbol as the last character in the file name. If you do not specify this parameter, PACK places the compressed data in *sourcefile* and modifies its name to contain the @ symbol.

/L

Indicates that *filename* is a list file. A list file is not compressed; it simply contains a listing of the names of the files that are to be compressed.

/H:headerpath\ or /H:headerfile or /H:headerpath\ headerfile

These parameters can be used separately or they can be paired.

/H:headerpath\

Specifies the destination path (drive letters are not permitted) to be placed in the header of the file that contains the compressed data. Unless this path is overridden with the UNPACK command, it will be the destination path when the file is uncompressed. *Headerpath* must end with a back slash (\).

/H:headerfile

Specifies the name of the file to be placed in the header of the compressed file. This file name will be used as the destination file for the uncompressed data and cannot be overridden.

If a header file name is not specified, PACK automatically uses *sourcefile* as the name of the file that is placed in the header of the compressed file.

/H:headerpath\ headerfile

Specifies that both a destination path and a destination file name are to be placed in the header of the file that has the compressed data.

/D:headerdate

Records the date in the header of the file that has the compressed data, and also in the destination file when it is uncompressed.

The date must follow the format /D:MM-DD-YYYY. For example: /D:08-20-1991 and /D:12-30-2010.

/T:headertime

Records the time in the header of the file that has the compressed data, and also in the destination file when it is uncompressed.

The time must follow the format /T:HH.MM. For example /T:02.06 and /T:14.54. Hour 00 represents 12 a.m. and hour 12 represents 12 p.m.

/C

Specifies that the current path be placed in the header of the file that contains the compressed data. When the UNPACK command is used, this path will be the destination path for the file that contains the uncompressed data.

You cannot use /C when the *headerpath* is used.

Note: The path and file-name information stored in the header of the file that contains the compressed data can be displayed by using the /SHOW option available with UNPACK. For information about the /SHOW option, see the UNPACK command in the online *OS/2 Command Reference* .

Creating a List File

To use a list file with PACK, you must first create a file that contains the names of the files you want to compress. You can give the list file any name. Following is an example of specifying a list file at the command line.

```
PACK DEVICE.LST DEVICE.DRV /L
```

The /L indicates that DEVICE.LST is a list file. If the list file is not in the working directory, you must specify the drive and path. Global file-name characters are not permitted in the list-file name. DEVICE.DRV is the destination file for the end-to-end-compressed data. (End-to-end compressed data is the data from each of the files contained in the list file. This data is stored in a contiguous format in the

destination file.)

The syntax used in the list file is similar to that used in the command line. The syntax for a single line in the list file follows:

```
sourcefile
  [/H:headerpath\
  |/H:headerfile
  |/H:headerpath\ headerfile]
  [/D:headerdate]
  [/T:headertime]
  [/C]
```

When using the list-file method (method 2), global file-name characters are not permitted in the source-file name. Notice also that "PACK" is excluded and *packedfile* is not permitted in the list file, because they were specified on the command line. You can include comments or blank lines by entering a semicolon as the first character of the line. An example of a list file follows.

```
;This is a comment
C:\OS2\COMMAND.COM
CONFIG.SYS /H:CONFIG.BAK /C
\OS2\INSTALL\DDINSTAL.EXE
 /H:\OS2\DDINSTAL.TMP
 /D:10-15-91 /T:11.45
```

Starting UNPACK

UNPACK restores a file of compressed data to its original size and copies it to a specified drive and path. To start the UNPACK command, type:

```
UNPACK sourcefile
[destinationdrive:] [destinationpath]
[/SHOW] [/N:singlefile]
[/V] [/F]
```

where:

sourcefile	Specifies the name of an existing file that contains compressed data. If this file contains one or more files of compressed data, UNPACK restores each file within the file.
destinationdrive:	Specifies the name of the drive where you want UNPACK to copy one or more restored files. When you specify a destination drive but not a path, UNPACK uses the path information stored in the header of the file that contains the compressed data.
destinationpath	Specifies the name of the directory (and its subdirectories) where you want UNPACK to copy one or more restored files. When specified, the destination path overrides the path information stored in the header of the file that contains the compressed data.
/SHOW	Displays the destination path and file-name information that are saved in the header of each file containing compressed data.
/N:singlefile	Extracts and uncompresses one file from a file that contains multiple files of compressed data.
/V	Verifies that sectors written to the target disk are recorded correctly. This parameter lets you know that critical data has been correctly recorded. This parameter causes UNPACK to run slower, because a check is made for each entry recorded on the disk.
/F	Specifies that files with extended attributes should not be unpacked or copied if the destination file system does not support extended attributes.

Resource Compiler

The OS/2 Resource Compiler (RC) is an application-development tool that lets you add application resources, such as message strings, pointers, menus, and dialog boxes, to the executable file of your application. The Resource Compiler is primarily intended to prepare data for OS/2 applications that use functions such as WinLoadString, WinLoadPointer, WinLoadMenu, and WinLoadDlg. These functions load resources from the executable file of your application or another specified executable file. The application can then use the loaded resources as needed.

The Resource Compiler and the resource functions let you quickly define and/or modify application resources without recompiling the application itself. That is, RC can modify the resources in an executable file at any time without affecting the rest of the file. This means that you can create custom applications from a single executable file - you just use RC to add the custom resources you need to each application.

The Resource Compiler is especially important for international applications because it lets you define all language-dependent data, such as message strings, as resources. Preparing the application for a new language is simply a matter of adding new resources to the existing executable file.

Using the Resource Compiler

The Resource Compiler (RC) compiles a resource script file to create a new file called a binary resource file.

The binary resource file can be added to the executable file of the application, replacing any existing resources in that file.

You can start RC in any of three ways.

- Compile and add a resource definition file to an executable file
- Compile a resource script file
- Add a binary resource file to an executable file

The RC command line has the following three basic forms:

```
rc resource-script-file [executable-file]

rc resource-file [executable-file]

rc -r  resource-script-file [resource-file]
```

Note: The third option does not add to the executable file.

The **resource-script-file** field must be the file name of the resource script file to be compiled. If the file is not in the current directory, you must provide a full path. If you provide a file name without specifying an extension, RC automatically appends the .RC extension to the name.

The **executable-file** field must be the name of the executable file to receive the compiled resources. This is a file having an extension of either .EXE or .DLL. If the file is not in the current directory, you must provide a full path. If you omit the executable-file field, RC adds the compiled resources to the executable file that has the same name as the resource script file but which has the .EXE file extension. If you specify the executable-file field but omit the extension, RC will append the .EXE extension. If this executable file does not exist, RC displays an error message.

The **-r** option directs RC to compile the resource script file without adding it to an executable file. You can use this option to prepare a binary resource file that you can add to an executable file at a later time. If you do not explicitly name a binary resource file along with the -r option, RC uses the same name as the resource script file but with the .RES extension.

The **resource-file** field must be the name of the binary resource file to be added to the executable file. If the binary resource file does not already exist, RC creates it; otherwise, RC replaces the existing file. If the file is not in the current directory, you must provide a full path. The binary resource file must have the .RES extension.

For example, to compile the resource script file EXAMPLE.RC, and add the result to the executable file EXAMPLE.EXE, use the following command:

```
rc example
```

You do not need to specify the .RC extension. RC creates the binary resource file EXAMPLE.RES and adds the compiled resource to the executable file EXAMPLE.EXE.

To compile the resource script file EXAMPLE.RC into a binary resource file without adding the resources to an executable file, use the following command:

```
rc -r example
```

The compiler creates the binary resource file EXAMPLE.RES. To create a binary resource file that has a name different from the resource script file, use the following command:

```
rc -r example newfile.res
```

To add the compiled resources in the binary resource file EXAMPLE.RES to an executable file, use the following command:

```
rc example.res
```

To specify the name of the executable file, if the name is different from the resource file, use the following command:

```
rc example.res newfile.exe
```

To add the compiled resources to a dynamic-link-library (.DLL) file, use the following command:

```
rc example.res dynalink.dll
```

Command-Line Options

The following options can be specified on the Resource Compiler command line:

-d <defname>[=<value>]	Define macro to preprocessor
-i <pathspec>	Include file path
-n	Suppress the display of the logo and the copyright information
-r	Create .res file
-p	Pack - 386 resources will not cross 64K boundaries.
-cp <cp>	Code page
-x[{1|2}]	Exepack - compress resources, using method 1 or 2.
-w2	Suppress the display of all warning and informational messages. Errors and fatal errors continue to be displayed.
-h (or -?)	Access Help

Leave a blank after the letter when using option -d, -i , or -cp. Uppercase or lowercase letters can be used.

If you omit the space after -d, the option works correctly although you receive an informational message stating that an invalid option was specified.

Explanation of Command-Line Options

The -d option is useful for passing conditional-compilation flags to the preprocessor. The <defname> is a sequence of letters, underscore symbols, and digits which does not begin with a digit. The <value> is a sequence of symbols which you want to substitute for the <defname> wherever it appears in the input script file. If you omit the =<value>, the <defname> will be set to the default value 1. For example, the option -d _3d is equivalent to including at the beginning of the input file this line:

#define _3d 1

You can use the -d option up to 8 times to define different macros from the command line.

The -i option defines paths for files to be included with the source file. The <pathspec> is any path where you want RC to search for files included by the preprocessor #include directive. The <pathspec> must not contain embedded blanks. To include more than one path, code the -i option once for each path. The preprocessor reads paths from the INCLUDE environment variable after reading the paths you provide with -i options.

The -r option will create in your current directory a binary resource file containing the resources you compile. The -r option takes no argument. The name given to this binary resource file will be the same as the name of the input resource script file except that the extension will be .RES instead of .RC. When you use -r, you do not bind resources to an executable file.

The -p option is used only when binding resources to an executable. It positions resources so that they do not cross 64K boundaries.

The -cp option is used to specify code page information for the resource script file to be compiled. The <codepage> is a numeric code page value. For a list of code page values, see the code page table under COUNTRYCODE in the online book *OS/2 Warp Control Program Programming Reference* .

The -x option is used only when binding resources to an executable. It causes resources to be compressed. These resources will be decompressed automatically when the resource is accessed.

The -x1 option causes Resource Compiler to use the compression algorithm that is compatible with OS/2 2.0, 2.1, and 2.11, as well as OS/2 3.0 and later.

The -x2 option causes Resource Compiler to use a compression algorithm that is compatible with OS/2 3.0 and later. The -x2 option will produce smaller executable files that can access resources faster.

-x with no number defaults to -x1.

The -n option (nologo) causes Resource Compiler to suppress the display of the logo and the copyright information.

The -w2 option causes Resource Compiler to suppress the display of all warning and informational messages. Errors and fatal errors continue to be displayed.

The -h and -? options cause Resource Compiler to display a summary of the available options and environment variables that it uses. When you use these options, Resource Compiler does not read any input files. Entering "RC" on the command line with no operands displays the same information.

Code Pages

In addition to -r, RC offers the **-cp** command-line option that lets you specify a code-page identifier. The syntax is as follows:

```
-cp codepage-id
```

The codepage-id field contains a valid code page. For a complete list of supported code pages, see the code page table under COUNTRYCODE in the online book *OS/2 Warp Control Program Programming Guide and Reference* .

Defining Constants

The -d option lets you define up to eight symbolic constants on the command line. The syntax is as follows:

```
-d defname[=value]
```

In the previous example, defname is a name, and value is an integer constant, or an expression. The -d option is useful for passing conditional-compilation flags to the RC preprocessor.

The following example specifies a Japanese code-page identifier and also defines two symbolic constants to be passed to the preprocessor as conditional-compilation flags.

```
rc -cp 942 -d DEBUG -d VERSION=2 example
```

Note: RC creates many temporary files and writes them to the directory indicated by the TMP or TEMP environment variable. If RC cannot write these temporary files to this directory, it writes them to the current directory.

Online Help

To display Resource Compiler help, type RC with no parameters, at a command prompt. The appropriate copyright statement will be displayed, along with a list of Resource Compiler options. You can also display this list by using the command-line options -h or -?.

```
Usage:  rc [<options>] <.RC input file> [<.EXE output file>]
   or:  rc [<options>] <.RES input file> [<.EXE output file>]
   or:  rc [<options>] -r <.RC input file> [<.RES output file>]

        -d <defname>    - Preprocessor define
        -D <defname>    - Preprocessor define
        -i <path>       - Include file path
        -r              - Create .res file
        -p              - Pack - 386 resources will not cross 64K boundaries
        -x[1|2]         - Exepack - Compress resources, using method 1 or 2
        -cp <cp>        - Code page
        -n              - Don't show logo
        -w2             - Suppress warnings
        -?              - Access Help
        -h              - Access Help

Environment variables:
        DBCS=<cp>
        TMP=<temporary file path>
        TEMP=<temporary file path>
        INCLUDE=<include file path>; ...
```

Note: Option -X2 will compress executable files that run only on OS/2 versions 3.0 and later.

Resource Script Files

This topic describes the resource script file used to define your application resources and explains how to compile the file and add the resources to your executable file.

Use the Resource Compiler to perform the following actions:

- Create a resource script file.
- Compile the file.
- Add the file to the executable file of your application (optional).

The following sections describe the resource script file and the RC program.

About Resource Statements

Resource statements have three basic forms:

- Single-line Statements
- Multiple-line Statements

- Directives

Each resource statement consists of one or more

- Keywords
- Numbers
- Expressions
- Character Strings
- Constants
- File Names

You combine these to define the resource type, identifier, and data.

Single-line Statements

Single-line statements consist of a keyword identifying the resource type, a number or character string which specifies the resource identifier, and a file name specifying the file containing the resource data. For example, this ICON statement defines an icon resource:

```
ICON 1 myicon.ico
```

The icon resource has the icon identifier 1. The file MYICON.ICO contains the icon data. The same example, using character strings for identifiers is shown below:

```
ICON "MyIcon" myicon.ico
```

Multiple-line Statements

Multiple-line statements consist of a keyword identifying the resource type, a number or character string which specifies the resource identifier, and, between the BEGIN and END keywords, additional resource statements that define the resource data. For example, this MENU statement defines a menu resource:

```
MENU 1
BEGIN
    MENUITEM "Alpha", 101
    MENUITEM "Beta",  102
END
```

The menu identifier is 1. The menu contains two MENUITEM statements that define the contents of the menu. In multiple-line statements such as DLGTEMPLATE and WINDOWTEMPLATE, RC allows any level of nested statements. For example, the DLGTEMPLATE and WINDOWTEMPLATE statements typically contain a single DIALOG or FRAME statement. These statements can contain any number of WINDOW and CONTROL statements; the WINDOW and CONTROL statements can contain additional WINDOW and CONTROL statements; and so on. The nested statements let you define controls and other child windows for the dialog boxes and windows. If a nested statement creates a child window or control, the parent and owner of the new window is the window created by the containing statement. (FRAME statements occasionally create frame controls whose parent and owner windows are not the same.)

Directives

Directives consist of the reserved character # in the first column of a line, followed by the directive keyword and any additional numbers, character strings, or file names. Valid directives include:

```
#define
#include
#if
```

```
#ifdef
#ifndef
#else
#elif
#endif
#undef
```

Keywords

Keywords are words that have a special meaning to the Resource Compiler. In a statement, keywords specify the resource type, the directive, the memory options, and the beginning and ending of nested statements. You can use keywords only as specified in the statement syntax.

Keywords, except for those specifying directives, can be any combination of uppercase and lowercase letters. Note that the left and right braces, { and }, are reserved characters. You can use them in place of the BEGIN and END keywords, respectively.

Numbers

Numbers are integers that represent coordinates, dimensions, styles, and other numeric data. You can specify numbers in decimal or in hexadecimal notation:

- Decimal numbers must contain decimal digits but can start with a minus sign (-) when they represent a negative number.
- Hexadecimal numbers must contain hexadecimal digits (uppercase or lowercase) and must start with the characters 0x or 0X. (These begin with the digit zero.)

The following example shows several numbers represented in decimal and hexadecimal notation:

Decimal	Hexadecimal
1	0x1
10	0xA
255	0xFF
-1	
65535	0xFFFF

Statements that create controls in dialog windows and menu items require that you specify an identifier for each control or menu item. Statements that create controls also require you to specify coordinates and dimensions. You specify identifiers, coordinates and dimensions using integers or simple expressions that evaluate to integers in the appropriate range. This enables you to specify identifiers, dimensions, and coordinates that are relative to those of the corresponding dialog window or menu. A resource identifier encoded as an expression must resolve to an unsigned integer, not a string.

The ranges specific to number types are listed in the following table.

	Signed Range	Unsigned Range	Strings
Identifiers	-32768 through 32767	1 through 65535	Yes
Coordinates	-32768 through 32767	0 through 65535	No
Dimensions	Not applicable	0 through 65535	No

Expressions

Expressions can be used whenever numbers can be used in a resource script file. An expression is a sequence of operators and operands that are evaluated to derive a numeric result. The result of an expression can be used whenever Resource Compiler expects a numeric value. The precedence and associativity of operators will be the same as the standard C programming language. Expressions may be nested within other expressions.

An example of operator precedence is illustrated by the following expression:

```
3 + 4 * 2
```

Because the multiplication operator (*) has a higher precedence than the addition operator (+), the computation 4 * 2 will be performed first.

An example of operator associativity is illustrated by the following expression:

```
8 / 2 * 3
```

Because both the division (/) and multiplication (*) operators are at the same precedence level, their left-to-right associative property causes the computation 8 / 2 to be performed first.

A Unary Expression consists of one operand and a unary operator.

```
(-1) is a unary expression.
```

A Binary Expression contains two operands separated by one operator.

```
(3 + 4)  is a binary arithmetic expression having evaluated value as 7.
(2 < 1)  is a binary logical expression having evaluated value as 0 (False).
(1 < 2)  is a binary logical expression having evaluated value as 1 (True).
(2 << 1) is a binary left shift expression having evaluated value as 4.
(4 >> 1) is a binary right shift expression having evaluated value as 2.
(15 % 4) is a binary modulus expression having evaluated value as 3 (remainder).
```

The following example specifies the resource ID as the number 12:

```
ICON 12 myfile.ico
```

This same resource ID can also be specified in the form of an expression, as shown here:

```
#define BASE_ID         10
#define ICON_OFFSET_ID  2
ICON (BASE_ID + ICON_OFFSET_ID) myfile.ico
```

The constants BASE_ID and ICON_OFFSET_ID will be replaced with their macro values. The result of the evaluated expression (10 + 2) will be 12 which is the same as in the first example. The number 10 and 2 are operands in the expression and the symbol plus (+) is the operator applied on operands 10 and 2.

Although the enclosing parentheses are not required, it is a good practice to enclose the expression in parentheses indicating that the whole expression constitutes a single parameter.

The following table mentions the valid operators that can be used in an expression.

Operator	Symbol	Type
add	+	binary
and	&	binary
and if	&&	binary
bitwise exclusive or	^	binary

bitwise negation	~	unary
bitwise negation	NOT	unary
divide	/	binary
equals	=	binary
greater than	>	binary
left shift	<<	binary
less than	<	binary
less than	<	binary
logical negation	!	unary
minus	-	binary
modulus	%	binary
multiply	*	binary
not equal	!=	binary
not less than	>=	binary
not greater than	<=	binary
or	|	binary
or else	||	binary
right shift	>>	binary

Note: The unary NOT operator is used only in the style field of resource statements. If operator NOT is encountered with the operands in an expression, it will be translated as a ~ character.

Any typecast with an operand will be ignored and a warning message will be generated. For example, the expression ((long)3 + (short)2) has typecast (long) with operand 3 and typecast (short) with operand 2. The typecasts have no effect on expression evaluation.

Character Strings

A character string, which may also be called a string constant or string literal, contains a sequence of characters or escape sequences enclosed in double quotation mark symbols. Character strings represent names, labels, titles, and messages. The meaning of each character value depends on the code page (character set) defined for the resource script file.

A string constant has the form:

```
>   "     character         "                                    >
          escape_sequence
```

Where:

character	The ASCII value of the character must be in the range 1 through 255.
escape_sequence	You can represent any member of the character set by an escape sequence. For example, you can use escape sequences to place such characters as tab, carriage return, and backspace in the character string. An escape sequence contains a backslash (\) character followed by one of the escape sequence characters: a, b, f, n, r, t, v, ', ", ?, \ or followed by an octal or hexadecimal number. A hexadecimal escape sequence contains an character x followed by one or more hexadecimal digits (0-9, A-F, a-f).

An octal escape sequence contains one or more octal digits (0-7). The value of the hexadecimal or octal number specifies the value of the desired character.

An escape sequence has the form:

```
>   \    escape_sequence_character                              ><
         octal_digits
         x hexadecimal_digits
```

Escape sequence	Character Represented
\a	Alert (bell)
\b	Backspace
\f	Form feed (new page)
\n	Newline
nnn	Octal character
\x*dd*	Hexadecimal character
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\\	Backslash

When you want the backslash to represent itself (rather than the beginning of an escape sequence), you must use a \\ (two backslashes) escape sequence.

If two consecutive strings appear on the same source line without any space between these two strings, the strings will be merged with an embedded quote.

```
/* Source Line 1 */ "String 1""String 2"
```

Here two strings are specified on the same line with no space between the strings. These strings will be merged as - String 1"String 2. Here the ending double quote of first string and begin double quote of second string creates the effect of embedded double quotes inside the merged string. These consecutive double quote characters will be translated as a single double quote character. This is the same representation as shown below:

```
/* Source Line 1 */ "String 1\"String 2"
```

Here an escape sequence \" is used to include double quote character inside the string. This is the recommended representation to include a double quote character inside the string.

If one or more consecutive strings are separated by one or more spaces (the end of source line is considered as space also), then strings will be concatenated to produce a single string. See the following example:

```
/* Source Line 1 */ "String 3:" "String 4",
/* Source Line 2 */ "String 5:"
/* Source Line 3 */ "String 6"
```

The two strings "String 3:" and "String 4" specified on source line 1 are on the same line separated with a single blank. These two strings will be concatenated as shown below:

```
String 3:String 4
```

The two strings "String 5:" and "String 6" specified on source lines 2 and 3 are on different source lines. These two strings will be concatenated the same way as with strings specified on source line 1.

```
String 5:String 6
```

Note that the comma had been specified after the "String 4" in source line 1 to stop the concatenation of strings on source lines 2 and 3. If that comma had not been specified, then all the strings on source lines 1, 2, and 3 would have been merged as shown in the following example:

```
String 3:String4String5:String6
```

If the string has no ending double quote mark at the end of the source line, then the physical end of source line will generate a newline character as part of the string data.

```
"This string on source line 1
is continued on source line 2"
```

In the above example, the newline character (hex 0A) will be inserted after character 1. This representation is equivalent to the following example.

```
"This string on source line 1\nis continued on source line 2"
```

Here the escape sequence \n is used to generate newline character (hex 0A) between two strings.

```
"String on Line 1
  String on Line 2"
```

In the above example, the second string on line 2 is specified with two leading spaces. These leading spaces will be part of the string data. If you do not want any leading space on the strings that span next lines, then start the strings in column 1 on the next lines. This representation is equivalent to the following example.

```
"String on Line 1\n  String on Line 2"

You can use an another approach like the following example:

"String on Line 1\n"
               " String on Line 2"
```

This approach is more readable. Here the concatenation operation will be performed on both strings without worry about the leading spaces before the string " String on Line 2". Leading spaces are provided inside the second string so you can place second string on the next line at the indented place. See the following STRINGTABLE statement example.

```
STRINGTABLE
{
   1, "The data on disk will be erased\n"
      "Are you sure (Y/N) \?"
}
```

You can use the escape sequence \n to represent a new-line character as part of the string. You can use the escape \\ to represent a backslash character as part of the string. You can use the escape \" to present the double quotation mark symbol as part of the string. You can include any ASCII character in a character string by specifying \xdd, where dd is the hexadecimal representation of an ASCII character. An error message is issued if an escape sequence is not recognized.

In addition, when character strings are used as resource identifiers additional rules apply:

- They must be enclosed in double quotation marks ("). If a double quotation mark is needed inside the string, it is encoded as two consecutive double quotation marks.
- They cannot contain any character larger than 0x7F.
- They must be delimited by whitespace, just as an integer ID is.
- Resources whose resource ID is compiled into a 16-bit value in the binary data area, such as MENUITEM with its menu-id field, or HELPSUBITEM with its child-window-id field, cannot use character strings for IDs.

- They can contain an embedded newline character by continuing the string on the following line without closing the string. When the input file represents newlines as 0x0D+0x0A or by 0x0D+0x0D+0x0A, the string is compiled with the sequence 0x0A to represent the newline.
- Duplicate string IDs are not permitted for resources of the same type. However, the same string resource identifier can be used to identify resources of different types.

  When the Resource Compiler is compiling a script file and encounters more than one resource of the same type having the same string ID, it will generate an error message and stop compiling the file. When the Resource Compiler is binding a .RES file and encounters more than one resource of the same type with the same string ID, it will generate a warning message and ignore the second resource identifier; only the first resource having the duplicated identifier will be bound to the file.

Constants

Constants are names that have been assigned values by using a #define directive or a -D command-line option. A constant can represent a number, a character string, or other data. Most resource statements in a resource script file use constants, and many use the constants defined in system header files. For this reason, you should consider using the #include directive to include the appropriate system header file in your resource script file.

File Names

File names are OS/2 file names. If the specified file is not in the current directory, you must specify the drive, directory, and file name.

Binary Resource Files

The binary resource file created by the Resource Compiler consists of one or more resource entries, each in the following form:

```
struct {
    UCHAR  fResType;
    USHORT usResType;
    union {
        struct {
            UCHAR  fResID;
            USHORT resid;
        };
        UCHAR resname[];
    };
    USHORT fsOptions;
    ULONG  cb;
    BYTE   bytes[1];
};
```

The fields in each entry have the following meanings:

fRestype — Specifies whether the resource-type identifier is a string or an integer. For OS/2, the resource type is always an integer and this field is set to 0xFF.

usResType — Specifies the resource-type identifier. This value is a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a character string. The following resource types are predefined:

RT_ACCELTABLE	Accelerator table
RT_BITMAP	Bitmap
RT_CHARTBL	Character table
RT_DIALOG	Dialog template
RT_DISPLAYINFO	Display information

RT_DLGINCLUDE	Dialog include-file name
RT_FKALONG	Long-form function-key area
RT_FKASHORT	Short-form function-key area
RT_FONT	Font
RT_FONTDIR	Font directory
RT_HELPSUBTABLE	Help subtable
RT_HELPTABLE	Help table
RT_KEYTBL	Key table
RT_MENU	Menu template
RT_MESSAGE	Error-message table
RT_POINTER	Mouse-pointer shape
RT_RCDATA	Binary data
RT_STRING	String table
RT_VKEYTBL	Virtual key table
RT_RESNAMES	String ID table

fResID — Specifies whether the resource identifier is a string or an integer. For the OS/2 operating system, this field is set to 0xFF to indicate that the resource identifier is an integer.

resid — Specifies the resource identifier. This value is an unsigned integer in the range of 1 through 65535.

resname — Specifies a string resource identifier as a sequence of characters ending with a 0x00 value.

fsOptions — Specifies the load and memory options. This value can be a combination of the following:

0x0010	MOVEABLE resource. If not given, the resource is FIXED.
0x0040	PRELOAD resource. If not given, the resource is LOADONCALL.
0x1000	DISCARDABLE resource.

cb — Specifies the size of the resource (in bytes).

bytes — Contains the resource.

Note: There is a size limitation of 65280 bytes for a binary resource file.

Statements and Directives

The following statements and directives are used by the Resource Compiler (RC):

ACCELTABLE Statement
ASSOCTABLE Statement
AUTOCHECKBOX Statement
AUTORADIOBUTTON Statement
BITMAP Statement
CHECKBOX Statement
CODEPAGE Statement
COMBOBOX Statement
CONTAINER Statement
CONTROL Statement
CTEXT Statement
CTLDATA Statement
DEFAULTICON Statement
define Directive
DEFPUSHBUTTON Statement
DIALOG Statement
DLGINCLUDE Statement
DLGTEMPLATE Statement
EDITTEXT Statement
elif Directive

ACCELTABLE Statement

Syntax:

```
ACCELTABLE acceltable-id [mem-option] [code-page]
BEGIN
key-value, command[, accelerator-options]...
    .
    .
    .
END
```

Description

The ACCELTABLE statement creates a table of accelerators for an application. An accelerator is a keystroke that gives the user a quick way to choose a command from a menu or carry out some other task. An accelerator table can be loaded when needed from the executable file by using the WinLoadAccelTable function.

You can provide any number of ACCELTABLE statements in a resource script file. Each statement must specify a unique table identifier. You can provide any number of accelerator definitions in an accelerator table; however, no two definitions in a table can specify the same key.

Each accelerator definition must specify a key value and command. The WinSetAccelTable function used in the application translates the accelerator keystroke into a WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message that has the corresponding command value. The message type depends on the accelerator-option field.

acceltable-id — Specifies the accelerator-table identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string. Each accelerator table in a resource script file must have a unique identifier.

mem-option — Specifies how the system manages the resource when it is in memory. This value must be one of the following:

Option	Meaning
FIXED	System keeps the resource at a fixed memory location.
MOVEABLE	System moves the resource as necessary to compact memory. This is the default option.
DISCARDABLE	System discards the resource if it is no longer needed.

code-page — Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.

key-value — Specifies the character, scan, or virtual-key code of the accelerator key. The meaning depends on the accelerator-options field. The key-value field must be a single character enclosed in double-quotation marks or an integer in the range 0 through 255. If you specify an integer, you must specify the CHAR, SCANCODE, or VIRTUALKEY accelerator option; otherwise, the default option is CHAR. Integers must be in decimal or hexadecimal notation.

command — Specifies the command value for the corresponding WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message. This value must be a signed integer in the range 0 through 65535, or a simple expression that evaluates to an integer in that range.

accelerator-options — Specifies the accelerator type. This value can be a combination of the following:

VIRTUALKEY	Specifies that the key-value field is a virtual-key code.
SCANCODE	Specifies that the key-value field is a keyboard scan code.
CHAR	Specifies that the key-value field is a character code.
SHIFT	Specifies that the user must press the Shift key and the key corresponding to the key-value field to generate the accelerator.
CONTROL	Specifies that the user must press the Ctrl key and the key corresponding to the key-value field to generate the accelerator.
ALT	Specifies that the user must press the Alt key and the key corresponding to the key-value field to generate the accelerator.
LONEKEY	Specifies that the user needs to press only the key corresponding to the key-value field to generate the accelerator.
SYSCOMMAND	Specifies that the accelerator translates to a WM_SYSCOMMAND message. If you do not include this option, the accelerator translates to a WM_COMMAND message.
HELP	Specifies that the accelerator translates to a WM_HELP message. If you do not include this option, the accelerator translates to a WM_COMMAND message.

Note: VIRTUALKEY, SCANCODE, and CHAR are mutually exclusive. SYSCOMMAND and HELP are also mutually exclusive.

Comments

If two accelerators use the same key with different Shift, Control, or ALT options, you should specify the more restrictive accelerator first in the table. For example, you should place Shift+Enter before Enter.

If you include the OS2.H header file, you can use the following constants to specify the accelerator options:

AF_ALT	AF_CHAR	AF_CONTROL
AF_HELP	AF_LONEKEY	AF_SCANCODE
AF_SHIFT	AF_SYSCOMMAND	AF_VIRTUALKEY

To combine these constants, you must use the bitwise OR (|) operator.

Example

This example creates an accelerator table whose accelerator-table identifier is 1. The table contains two accelerators: Ctrl+S and Ctrl+G.

These accelerators generate WM_COMMAND messages with values of 101 and 102, respectively, when the user presses the corresponding keys.

```
ACCELTABLE 1
BEGIN
    "S", 101, CONTROL
    "G", 102, CONTROL
END
```

ASSOCTABLE Statement

Syntax:

```
ASSOCTABLE assoctable-id [load-option][mem-option] [code-page]
BEGIN
association-name, file-match-string[, extended-attribute-flag]
   [, icon-filename]
  .
  .
  .
END
```

Description

The ASSOCTABLE statement defines a file-association table for an application. This table associates the data files that an application creates with the executable file of the application. When the user selects one of these data files from File Manager, the associated application begins executing.

A file-association table can also associate icons with the data files that an application creates. The shell uses these icons to identify the data files graphically. Because a file-association table associates icons by file type, all data files having the same file type have the same icon.

You can provide any number of ASSOCTABLE statements in a resource script file, but each statement must specify a unique assoctable-id value. The file-association tables are written not only to the resources within your executable file, but also to the .ASSOC extended attribute. However, only the last file-association table specified in the resource script file is actually written to the extended attribute.

assoctable-id	Specifies the association-table identifier. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. Character strings cannot be used as resource identifiers for this statement.
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:
PRELOAD	System loads the resource when the application starts.
LOADONCALL	System loads the resource when the application calls the DosGetResource or DosGetResource2 function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one of the following:
FIXED	System keeps the resource at a fixed memory location.
MOVEABLE	System moves the resource as necessary to compact memory. This is the default option.
DISCARDABLE	System discards the resource if it is no longer needed.
code-page	Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.
association-name	Specifies the name of the file type the application recognizes. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the name, you must include the double quotation mark twice.
file-match-string	Specifies the file-matching string of a particular type of data file that the application creates. This field must contain zero or more characters enclosed in double quotation marks. You can only use characters that are valid in OS/2 file names and extensions and the OS/2 wildcard characters question mark (?) and asterisk (*).
extended-attribute-flag	Specifies the extended-attribute options. This value can be a combination of the following:
EAF_DEFAULTOWNER	Specifies that the application

		containing the file-association table starts when the user selects any file matching the file-match-string field from File Manager.
	EAF_REUSEICON	Specifies that the icon defined in the previous entry of the file-association table is used as the icon for the current data-file type.
	EAF_UNCHANGEABLE	Specifies that the entry should not be edited.
icon-filename	Specifies the name of the file containing an icon. File Manager uses this icon to represent all application-created data files matching the file-match-string field. The file must be in the current directory.	

AUTOCHECKBOX Statement

Syntax:

```
AUTOCHECKBOX text, id, x, y, width, height[, style]
```

The AUTOCHECKBOX statement creates an automatic-check-box control. The control is a small rectangle (check box) that contains an X when the user selects it. The specified text is displayed to the right of the check box. An X appears in the square when the user first selects the control and disappears the next time the user selects it. The AUTOCHECKBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify the style, the default style is BS_AUTOCHECKBOX and WS_TABSTOP.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates an automatic-check-box control that is labeled "Italic."

```
AUTOCHECKBOX "Italic", 101, 10, 10, 100, 100
```

AUTORADIOBUTTON Statement

Syntax:

```
AUTORADIOBUTTON text, id, x, y, width, height[, style]
```

The AUTORADIOBUTTON statement creates an automatic-radio-button control. This control is a small circle with the given text displayed to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control also removes the selection from any other automatic-radio-button controls in the same group. When the user selects the button again, the control removes the highlight before sending a message. The AUTORADIOBUTTON statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_AUTORADIOBUTTON.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates an automatic-radio-button control that is labeled "Italic."

```
AUTORADIOBUTTON "Italic", 101, 10, 10, 24, 50
```

BITMAP Statement

Syntax:

```
BITMAP bitmap-id [load-option] [mem-option] [codepage] filename
```

The BITMAP statement defines a bit map resource for an application. A bit map resource, typically created using the Icon Editor, is a custom bit map that an application uses in its display or as an item in a menu. The BITMAP statement copies the bit-map resource from the file specified in the filename field and adds it to the application's other resources. A bit-map resource can be loaded from the executable file when needed by using the GpiLoadBitmap function.

You can provide any number of BITMAP statements in a resource script file, but each statement must specify a unique bitmap-id value.

bitmap-id	Specifies the bit-map-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the GpiLoadBitmap function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one of the following:	
	FIXED	System keeps the resource at a fixed memory

		location.
	MOVEABLE	System moves the resource as necessary to compact memory. This is the default option.
	DISCARDABLE	System discards the resource if it is no longer needed.
codepage		Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.
filename		Specifies the name of the file containing the icon resource. If the file is not in the current directory, filename must be preceded by a full path.

Example

This example defines a bit map whose bit-map identifier is 12. The bit-map resource is copied from the file CUSTOM.BMP.

```
BITMAP 12 custom.bmp
```

CHECKBOX Statement

Syntax:

```
CHECKBOX text, id, x, y, width, height[, style]
```

The CHECKBOX statement creates a check-box control. The control is a small rectangle (check box) that has the specified text displayed to the right. The control highlights the rectangle and sends a message to its parent window when the user selects the control. The CHECKBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_CHECKBOX and WS_TABSTOP.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a check-box control that is labeled "Italic."

```
CHECKBOX "Italic", 101, 10, 10, 100, 100
```

CODEPAGE Statement

Syntax:

```
CODEPAGE codepage-id
```

The CODEPAGE statement sets the code page for all subsequent resources. The code page specifies the character set used for characters in the resource.

If the CODEPAGE statement is not given in a resource script file, RC uses the code page set up for the individual system. If more than one CODEPAGE statement is given in the file, each CODEPAGE statement applies to the resource statements between it and the next CODEPAGE statement.

codepage-id	Identifies the code page to be used for subsequent resources. For a complete list of supported code pages, refer to the "COUNTRYCODE" section of the *Control Program Programming Guide and Reference* .

Comments

You may also specify a code page by placing a code-page identifier in the load-options or memory-options field of any RC statement that uses those fields.

Example

In this example, the code page for the character-string resources is set to Portuguese (860).

```
CODEPAGE 860

STRINGTABLE
BEGIN
    1 "Filename not found"
    2 "Cannot open file for reading"
END
```

COMBOBOX Statement

Syntax:

```
COMBOBOX text, id, x, y, width, height[, style]
```

The COMBOBOX statement creates a combination-box control. This control combines a list-box control with an entry-field control. It allows the user to place the selected item from a list box into an entry field.

The COMBOBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_COMBOBOX. If you do not specify a style, the default style is CBS_SIMPLE, WS_GROUP, WS_TABSTOP, and WS_VISIBLE.

text	Specifies text that is displayed in the entry field of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_COMBOBOX. You can use

the bitwise OR (|) operator to combine styles.

Example

This example creates a combination-box control.

```
  COMBOBOX "", 101, 10, 10, 24, 50
```

CONTAINER Statement

Syntax:

```
  CONTAINER  id, x, y, width, height [,style]
```

The CONTAINER statement creates a container control within a dialog window. The container control is a visual component that holds objects. The CONTAINER statement defines the identifier, position, dimensions, and attributes of a container control. The predefined class for this control is WC_CONTAINER. If you do not specify a style, the default style is WS_TABSTOP, WS_VISIBLE, and CCS_SINGLESEL.

id	Specifies the control identifier. This value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window containing the container control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window containing the container control.
width	Specifies the width of the control. This value is any integer 0 through 65535, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value is any integer 0 through 65535, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_CONTAINER. Use the bitwise OR (|) operator to combine styles.

Comments

A CONTAINER statement is only used in a DIALOG or WINDOW statement.

Example

This example creates a container control at position (30,30) within the dialog window. The container has a width of 70 character units and a height of 25 character units. Its resource identifier is 301. The default style CCS_SINGLESEL has been overridden by the style specification CCS_MULTIPLESEL. The default styles WS_TABSTOP and WS_GROUP are both in effect, though only the latter is specified.

```
#define IDC_CONTAINER    301
#define IDD_CONTAINERDLG 504
DIALOG "Container", IDD_CONTAINERDLG, 23, 6, 120, 280, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
     CONTAINER   IDC_CONTAINER, 30, 30, 70, 200, CCS_MULTIPLESEL |
                       WS_GROUP
  END
```

CONTROL Statement

Syntax:

```
  CONTROL text, id, x, y, width, height, class[, style]
          [data-definitions]
```

```
[ BEGIN
control-definition
    .
    .
    .
END ]
```

The CONTROL statement defines a control as belonging to the specified class. The statement defines the position and dimensions of the control within the parent window, as well as the control style. The CONTROL statement is most often used in a DIALOG or WINDOW statement.

Typically, several CONTROL statements are used in each DIALOG statement, and each CONTROL statement must have a unique identifier value. The optional BEGIN and END statements enclose any CONTROL statements that may be given with the control. CONTROL statements given in this manner represent child windows belonging to the control created by the CONTROL statement.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. In the appropriate styles, a tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character. When the style field for this control includes the style SS_BITMAP, the text field should be written as a number equal to the resource identifier of the bitmap to be loaded.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the parent window.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the parent window.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in 1/8-character units.
class	Specifies the control class. This value can be one of the control classes specified in the "Control Classes" table, in the Presentation Manager Programming Reference, or the name of the control class, enclosed in double quotation marks.
style	Specifies the control style. This value can be a combination of control styles. You can use the bitwise OR (|) operator to combine styles.
data-definitions	Specifies a CTLDATA and/or PRESPARAMS statement. These statements define control and presentation data for the control. For more information, see CTLDATA Statement and PRESPARAMS Statement.
control-definition	Specifies a CONTROL statement or any one of several predefined control statements. These statements define the style, position, and dimensions of controls in the control.

Comments

The CONTROL statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. But typically, a CONTROL statement contains no such statements.

Example

This example creates a pushbutton control with the WS_TABSTOP and WS_VISIBLE styles.

```
CONTROL "OK", 101, 10, 10, 20, 50, WC_BUTTON, BS_PUSHBUTTON |
                                               WS_TABSTOP    |
                                               WS_VISIBLE
```

CTEXT Statement

Syntax:

```
CTEXT text, id, x, y, width, height[, style]
```

The CTEXT statement creates a centered-text control. The control is a simple rectangle displaying the given text centered in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The CTEXT statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of the control. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, DT_CENTER, and WS_GROUP.

text	Specifies text that is centered in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a centered-text control that is labeled "Filename."

```
CTEXT "Filename", 101, 10, 10, 100, 100
```

CTLDATA Statement

Syntax:

```
CTLDATA word-value[, word-value][...]

CTLDATA string

CTLDATA MENU
BEGIN
menuitem-definition
    .
    .
    .
END
```

The CTLDATA statement defines control data for a custom dialog box, window, or control. The statement has three basic forms to permit specifying a menu or specifying data in words or characters. The data can be in any format, since only your window procedure will use it. The window procedure of the dialog box, window, or control receives this data when the item is created. It is up to the window procedure to process the data.

word-value	Specifies a 16-bit value. This value must be a signed integer in the range -32768 through 32767 or an expression that evaluates in that range.
string	Specifies a string of 8-bit characters. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must include the double quotation mark twice.

menuitem-definition	Specifies a MENUITEM or SUBMENU statement. These statements define the individual commands or submenus in the given menu. For details about these statements, see MENUITEM Statement and SUBMENU Statement.

Comments

CTLDATA is often used to supply data that controls the subsequent operation of the custom window. For example, the CTLDATA statement may contain extended style bits - that is, style bits designed specifically for your customized window.

You should reserve the CTLDATA statement for window classes that you create yourself.

Example

This example creates a menu for the window created with the WINDOW statement.

```
WINDOWTEMPLATE 1
BEGIN
    WINDOW "Sample", 1, 0, 0, 100, 100, "MYCLASS", 0, FCF_STANDARD
    CTLDATA MENU
    BEGIN
        MENUITEM "Exit", 101
    END
END
```

DEFAULTICON Statement

Syntax:

```
DEFAULTICON filename.ico
```

This statement installs the named icon file definition under the ICON Extended Attribute of the program file. An icon with an icon-id of 1 is the default unless you supply a different icon.

Example

```
DEFAULTICON myicon.ico
```

---

define Directive

Syntax:

```
define name value
```

The define directive assigns the given value to the specified name. All subsequent occurrences of the name are replaced by the value.

name	Specifies the name to be defined. This name can be any combination of letters, digits, or underscore characters which does not begin with a digit.
value	Specifies any integer, character string, or line of text. This value can contain another defined name, which creates a level of nested defines. You are limited to 64 levels of nested defines.

Example

This example assigns values to the names "NONZERO" and "USERCLASS".

```
#define     NONZERO     1
```

```
#define     USERCLASS   "MyControlClass"
```

DEFPUSHBUTTON Statement

Syntax:

```
DEFPUSHBUTTON text, id, x, y, width, height[, style]
```

The DEFPUSHBUTTON statement creates a default pushbutton control. The control is a round-cornered rectangle containing the given text. The rectangle has a bold outline to represent that it is the default response for the user. The control sends a message to its parent window when the user chooses the control. The DEFPUSHBUTTON statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of the control. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_PUSHBUTTON, BS_DEFAULT, and WS_TABSTOP.

text	Specifies text that is centered in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a default pushbutton control that is labeled "Cancel."

```
DEFPUSHBUTTON "Cancel", 101, 10, 10, 24, 50
```

DIALOG Statement

Syntax:

```
DIALOG text, id, x, y, width, height[, style[, framectl]]
       [data-definitions]
BEGIN
control-definition
    .
    .
    .
END
```

The DIALOG statement defines a window that an application can use to create dialog boxes. The statement defines the position and

dimensions of the dialog box on the screen, as well as the dialog-box style. The DIALOG statement is most often used in a DLGTEMPLATE statement.

Typically, you use only one DIALOG statement in each DLGTEMPLATE statement, and the DIALOG statement contains at least one control definition.

text	Specifies the dialog-box title if the style specifies a title bar. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the title, you must include the double quotation mark twice.
id	Specifies the dialog-box identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the dialog box. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in dialog units, but its exact meaning depends on the dialog style. See the "Comments" section for details.
y	Specifies the y-coordinate of the lower-left corner of the dialog box. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in dialog units, but its exact meaning depends on the dialog style. See the "Comments" section for details.
width	Specifies the width of the dialog box. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in n-character units.
height	Specifies the height of the dialog box. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in 1/8-character units.
style	Specifies the dialog-box style. This value can be any of the window, dialog-box, or frame styles. You can use the bitwise OR (|) operator to combine styles.
framectl	Specifies the styles for frame controls belonging to the dialog box. This value can be any of the frame-control styles specified in the "Frame-Control Flags" table in the Presentation Manager Programming Reference. You can use the bitwise OR (|) operator to combine styles.
data-definitions	Specifies a CTLDATA and/or PRESPARAMS statement. These statements define control and presentation data for the dialog box. For more information, see CTLDATA Statement and PRESPARAMS Statement.
control-definition	Specifies a CONTROL statement or any one of several predefined control statements. These statements define the style, position, and dimensions of controls in the dialog box.

Comments

The exact meaning of the coordinates depends on the style defined by the style field. For dialog boxes with FS_SCREENALIGN style, the coordinates are relative to the origin of the display screen. For dialog boxes with the style FS_MOUSEALIGN, the coordinates are relative to the position of the mouse pointer at the time the dialog box is created. For all other dialog boxes, the coordinates are relative to the origin of the parent window.

The DIALOG statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a DIALOG statement contains one or more CONTROL statements.

Example

This example creates a dialog box that is labeled "Disk Error."

```
DLGTEMPLATE 1
BEGIN
    DIALOG  "Disk Error", 100, 10, 10, 300, 110
    BEGIN
        CTEXT "Select One:", 1, 10, 80, 280, 12
        RADIOBUTTON "Retry", 2, 75, 50, 60, 12
        RADIOBUTTON "Abort", 3, 75, 30, 60, 12
        RADIOBUTTON "Ignore", 4, 75, 10, 60, 12
    END
END
```

DLGINCLUDE Statement

Syntax:

```
  DLGINCLUDE id filename
```

The DLGINCLUDE statement adds the specified file to the resource file. The DLGINCLUDE statement is typically used to let the application access the definitions file for the dialog box with the corresponding identifier. The file named by filename must contain the define directives used by the dialog box.

You can provide any number of DLGINCLUDE statements in a resource script file, but each must have a unique identifier.

id	Specifies the dialog-box identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.
filename	Specifies the name of the file containing the define directives. If the file is not in the current directory, filename must be preceded by a full path.

Example

This example includes the name of the definition file dlgdef.h. The dialog-box identifier is 5.

```
DLGINCLUDE 5 \\INCLUDE\\DLGREF.H
```

DLGTEMPLATE Statement

Syntax:

```
DLGTEMPLATE dialog-id  [load-option] [mem-option] [codepage]
BEGIN
dialog-definition
    .
    .
    .
END
```

The DLGTEMPLATE statement creates a dialog-box template. A dialog-box template consists of a series of statements that define the identifier, load and memory options, dialog-box dimensions, and controls in the dialog box. The dialog-box template can be loaded from the executable file by using the WinLoadDlg function.

You can provide any number of dialog-box templates in a resource script file, but each template must have a unique dialog-id value.

dialog-id	Specifies the dialog-box identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the WinLoadDlg function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory.
	DISCARDABLE	System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage	Specifies a code-page value. For a list of valid code pages see CODEPAGE Statement.	
dialog-definition	Specifies a DIALOG statement. The statement defines the dimensions and style of the given dialog box. For details about the statement, see DIALOG Statement.	

Comments

A DLGTEMPLATE statement can actually contain DIALOG, CONTROL, and WINDOW statements. Typically, you include only one DIALOG

statement.

Example

This example uses a DLGTEMPLATE statement to create a dialog box.

```
DLGTEMPLATE ID_GETTIMER
BEGIN
    DIALOG "Timer", 1, 10, 10, 100, 40
    BEGIN
        LTEXT "Time (0 - 15):", 4, 8, 24, 72, 12
        ENTRYFIELD "0", ID_TIME, 80, 28, 16, 8, ES_MARGIN
        DEFPUSHBUTTON "Enter", ID_TIMEOK, 10, 6, 36, 12
        PUSHBUTTON "Cancel", ID_TIMECANCEL, 52, 6, 40, 12
    END
END
```

EDITTEXT Statement

Syntax:

```
  EDITTEXT text, id, x, y, width, height [,style]
```

The EDITTEXT statement creates an entry-field control. This control is a rectangle in which the user can type and edit text. The control displays a pointer when the user selects the control. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. By using the mouse or the DIRECTION keys, the user can select the character or characters to delete or select the place to insert new characters.

The EDITTEXT statement defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_ENTRYFIELD. If you do not specify a style, the default style is ES_AUTOSCROLL and WS_TABSTOP.

text	Specifies text that is displayed in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value is a signed integer -32768 through 32767, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. This value is a signed integer -32768 through 32767, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. This value is any integer 0 through 65535, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value is any integer 0 through 65535, or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_ENTRYFIELD. You can use the bitwise OR (|) operator to combine styles.

Comments

The EDITTEXT control statement is identical to the ENTRYFIELD control statement.

Use the EDITTEXT statement only in a DIALOG or WINDOW statement.

Example

This example creates an entry-field control that is not labeled.

```
   EDITTEXT "", 101, 10, 10, 24, 50
```

---

elif Directive

Syntax:

```
elif constant-expression
```

The elif directive marks an optional clause of a conditional-compilation block defined by a ifdef, ifndef, or if directive. The directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, elif directs the compiler to continue processing statements up to the next endif, else, or elif directive and then skip to the statement after endif. If the constant expression is zero, elif directs the compiler to skip to the next endif, else, or elif directive. You can use any number of elif directives in a conditional block.

constant-expression	Specifies the expression to be checked. This value is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

In this example, elif directs the compiler to process the second BITMAP statement only if the value assigned to the name "Version" is less than 7. The elif directive itself is processed only if Version is greater than or equal to 3.

```
#if Version < 3
BITMAP 1 errbox.bmp
#elif Version < 7
BITMAP 1 userbox.bmp
#endif
```

---

else Directive

Syntax:

```
else
```

The else directive marks an optional clause of a conditional-compilation block defined by a ifdef, ifndef, or if directive. The else directive must be the last directive before the endif directive.

This directive has no arguments.

Example

This example compiles the second BITMAP statement only if the name "DEBUG" is not defined.

```
#ifdef DEBUG
    BITMAP 1 errbox.bmp
#else
    BITMAP 1 userbox.bmp
#endif
```

---

endif Directive

Syntax:

```
endif
```

The endif directive marks the end of a conditional-compilation block defined by a ifdef directive. One endif is required for each if, ifdef, or ifndef directive.

This directive has no arguments.

ENTRYFIELD Statement

Syntax:

```
ENTRYFIELD text, id, x, y, width, height , [style]
```

The ENTRYFIELD statement creates an entry-field control. This control is a rectangle in which the user can type and edit text. The control displays a pointer when the user selects the control. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. By using the mouse or the DIRECTION keys, the user can select the character or characters to delete or select the place to insert new characters. The ENTRYFIELD statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_ENTRYFIELD. If you do not specify a style, the default style is ES_AUTOSCROLL and WS_TABSTOP.

text	Specifies text that is displayed in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_ENTRYFIELD. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates an entry-field control that is not labeled.

```
ENTRYFIELD "", 101, 10, 10, 24, 50
```

FONT Statement

Syntax:

```
FONT font-id  [load-option] [mem-option] [codepage]  filename
```

The FONT statement defines a font resource for an application. A font resource, typically created by using the OS/2 Font Editor, is a bit map defining the shape of the individual characters in a character set. The FONT statement copies the font resource from the file specified in the filename field and adds it to the other resources of the application. A font resource can be loaded from the executable file when needed by using the GpiLoadFonts function.

You can provide any number of FONT statements in a resource script file, but each statement must specify a unique font-id value.

font-id	Specifies the font-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. Character strings cannot be used as resource identifiers for this statement.
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following: PRELOAD — System loads the resource when the application starts. LOADONCALL — System loads the resource when the application calls the GpiLoadFonts function. This is the default option.
mem-option	Specifies how the system manages the resource when it is i memory. This value must be one or more of the following: FIXED — System keeps the resource at a fixed memory location. MOVEABLE — System moves the resource as necessary to compact memory. DISCARDABLE — System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage	Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.
filename	Specifies the name of the file containing the font resource. If the file is not in the current directory, filename must be preceded by a full path.

Example

This example defines a font whose font identifier is 5. The font resource is copied from the file cmroman.fon.

```
FONT 5 cmroman.fon
```

FRAME Statement

Syntax:

```
FRAME text, id, x, y, width, height[, style[, framectl]]
      data-definitions
[ BEGIN
window-definition
    .
    .
    .
END ]
```

The FRAME statement defines a frame window. The statement defines the title, identifier, position, and dimensions of the frame window, as well as the window style. The FRAME statement is most often used in a WINDOWTEMPLATE statement, and typically, only one FRAME statement is used. The FRAME statement, in turn, typically contains at least one WINDOW statement that defines the client window belonging to the frame window.

The frame window has no default style. You must use the **framectl** field to define additional frame controls, such as a title bar and system menu, to be created when the frame window is created. If the text field is not empty, the statement automatically adds a title-bar control to the frame window, whether or not you specify the FCF_TITLEBAR style. Frame controls are given default styles and control identifiers depending on their class. For example, a title-bar control receives the identifier FID_TITLEBAR.

text	Specifies the title of the frame window. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the name, you must include the double quotation mark twice.
id	Specifies the window identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the window. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified window.
y	Specifies the y-coordinate of the lower-left corner of the window. This value must be a signed integer in the

	range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified window.
width	Specifies the width of the window. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the window. This value must be a integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the frame and window styles. This value can be a combination of frame styles. You can use the bitwise OR (|) operator to combine styles.
framectl	Specifies the styles of frame controls belonging to the frame window. This value can be a combination of the styles specified in the "Frame-Control Styles" table in the Presentation Manager Programmers Reference. You can use the bitwise OR (|) operator to combine styles.
data-definitions	Specifies a CTLDATA and/or PRESPARAMS statement. These statements define control and presentation data for the frame window. For more information, see CTLDATA Statement and PRESPARAMS Statement.
window-definition	Specifies a WINDOW statement or any one of several predefined control statements. These statements define the style, position, and dimensions of windows or controls in the frame window.

Comments

The FRAME statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a FRAME statement contains one WINDOW statement.

Example

This example creates a standard frame window, with a title bar, a system menu, minimize and maximize boxes, and a vertical scroll bar. The FRAME statement contains a WINDOW statement defining the client window belonging to the frame window.

```
WINDOWTEMPLATE 1
BEGIN
    FRAME "My Window", 1, 10, 10, 320, 130, 0,
            FCF_STANDARD | FCF_VERTSCROLL
    BEGIN
        WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass"
    END
END
```

GROUPBOX Statement

Syntax:

```
  GROUPBOX text, id, x, y, width, height [, style]
```

The GROUPBOX statement creates a group-box control. The control is a rectangle that groups other controls together. The controls are grouped by drawing a border around them and displaying the given text in the upper-left corner. The GROUPBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_GROUPBOX and WS_TABSTOP.

text	Specifies text that appears in the upper-left corner of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the

bitwise OR (|) operator to combine styles.

Example

This example creates a group-box control that is labeled "Options."

```
GROUPBOX "Options", 101, 10, 10, 100, 100
```

HELPITEM Statement

Syntax:

```
HELPITEM application-window-id, help-subtable-id, extended-helppanel-id
```

The HELPITEM statement defines the help items in a help table. The statement, permitted only in a HELPTABLE statement, specifies the resource identifier of an application window for which help is provided, and the resource identifiers of the help subtable and extended help panel associated with the application window.

You can provide any number of HELPITEM statements in a HELPTABLE statement. You should provide one HELPITEM statement for each application window for which help is provided.

application-window-id	Specifies the resource identifier of an application window for which help is provided. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
help-subtable-id	Specifies the resource identifier of the help subtable associated with the specified application window. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
extended-helppanel-id	Specifies the resource identifier of the extended help panel associated with the specified application window. This value must be an integer in the range 0 through 65535. However, in IPF a panel-id must be an integer in the range of 0 to 64000.

Example

This example defines a help item that associates a help subtable called IDSUB_FILEMENU and an extended help panel called IDEXT_APPHLP with an application window called IDWIN_FILEMENU.

```
HELPITEM IDWIN_FILEMENU, IDSUB_FILEMENU, IDEXT_APPHLP
```

HELPSUBITEM Statement

Syntax:

```
HELPSUBITEM child-window-id, helppanel-id [ , integer...]
```

The HELPSUBITEM statement defines the help subitems in a help subtable. This statement, which is permitted only in a HELPSUBTABLE statement, specifies the identifier of a child window for which help is provided, the identifier of the help panel associated with the child window, and one or more optional, application-defined integers.

You can provide any number of HELPSUBITEM statements in a HELPSUBTABLE statement. You should provide one HELPSUBITEM statement for each child window for which help is provided.

child-window-id
: Specifies the resource identifier of the child window for which help is provided. Character strings cannot be used as resource identifiers for this statement.

helppanel-id
: Specifies the resource identifier of the help panel associated with the specified child window.

integer
: Specifies optional, application-defined integers. If you use this field, you must include the SUBITEMSIZE statement in the help subtable to specify the size, in words, of each help subitem in the help subtable. For details about this statement, see SUBITEMSIZE Statement.

Example

This example defines a help subitem that associates a child window called IDCLD_FILEMENU with a help panel called IDHP_FILEMENU.

```
HELPSUBITEM IDCLD_FILEMENU, IDHP_FILEMENU
```

HELPSUBTABLE Statement

Syntax:

```
  HELPSUBTABLE helpsubtable-id
    SUBITEMSIZE size
  BEGIN
  helpsubitem-definition
      .
      .
      .
  END
```

The HELPSUBTABLE statement defines the contents of a help-subtable resource. A help-subtable resource contains a help-subitem entry for each item that can be selected in an application window. Each of these items should be a child window of the application window specified in the help-table resource. The help subtable should contain a help subitem for each control, child window, and menu item in the application window.

You can provide any number of HELPSUBTABLE statements in a resource script file, but each statement must specify a unique helpsubtable-id value. You can also provide any number of helpsubitem-definition statements in the help subtable. These specify the child window for which help is provided, the help panel containing the help text for the child window, and one or more application-defined integers.

If you include optional integers in the helpsubitem-definition statements, you must also include a SUBITEMSIZE statement to specify the size, in words, of each help subitem. All help subitems in a help subtable must be the same size. The default size is two words per help subitem. (No SUBITEMSIZE statement is needed if you do not include optional integers in the helpsubitem-definition statement.)

helpsubtable-id
: Specifies the resource identifier of the help subtable. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. Character strings cannot be used as resource identifiers for this statement.

helpsubitem-definition
: Specifies a HELPSUBITEM statement. A HELPSUBITEM statement specifies a child window, the help panel associated with the child window, and one or more optional, application-defined integers. For details about this statement, see HELPSUBITEM Statement.

Example

This example creates a help-subtable resource whose help-subtable identifier is IDSUB_FILEMENU. Each HELPSUBITEM statement specifies a child window and a help panel.

```
HELPSUBTABLE IDSUB_FILEMENU
BEGIN
    HELPSUBITEM IDCLD_OPEN, IDPNL_OPEN
    HELPSUBITEM IDCLD_SAVE, IDPNL_SAVE
END
```

HELPTABLE Statement

Syntax:

```
HELPTABLE helptable-id
BEGIN
helpitem-definition
    .
    .
    .
END
```

The HELPTABLE statement defines the contents of a help-table resource. A help-table resource contains a help-item entry for each application window, dialog box, and message box for which help is provided.

You can provide any number of HELPTABLE statements in a resource script file, but each statement must specify a unique helptable-id value. You can also provide any number of helpitem-definition statements in the help table. These specify the application windows for which help is provided, the help subtables associated with each application window, and the extended help panels associated with each application window.

helptable-id	Specifies the resource identifier of the help table. This value must be an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. Character strings cannot be used as resource identifiers for this statement.
helpitem-definition	Specifies a HELPITEM statement. A HELPITEM statement specifies an application window and the associated help subtable and extended help panel. For details about this statement, see HELPITEM Statement.

Example

This example creates a help-table resource whose help-table identifier is 1. Each HELPITEM statement specifies an application window, a help subtable, and an extended help panel.

```
HELPTABLE 1
BEGIN
    HELPITEM IDWIN_FILEMENU, IDSUB_FILEMENU, IDEXT_APPHLP
    HELPITEM IDWIN_EDITMENU, IDSUB_EDITMENU, IDEXT_APPHLP
END
```

ICON Statement (Resource)

Syntax:

```
ICON icon-id  [load-option] [ mem-option] [codepage] filename
```

This form of the ICON statement defines an icon resource for an application. An icon resource, typically created by using Icon Editor, is a bit map defining the shape of the icon to be used for a given application. The ICON statement copies the icon resource from the file specified in the filename field and adds it to the application's other resources. An icon resource can be loaded when creating a window by using the WinCreateStdWindow function with the FS_ICON style.

You can provide any number of ICON statements in a resource script file, but each statement must specify a unique icon-id value.

icon-id	Specifies the icon-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string. An icon-id of 1 has a special meaning; for details, see the "Comment" section.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the WinCreateStdWindow function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:	
	FIXED	System keeps the resource at a fixed memory

		location.
	MOVEABLE	System moves the resource as necessary to compact memory.
	DISCARDABLE	System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage		Specifies a code page value. For a list of valid code pages see CODEPAGE Statement.
filename		Specifies the name of the file containing the icon resource. If the file is not in the current directory, filename must be preceded by a full path.

Comments

An icon with an icon-id of 1 is the default icon. The RC program writes the icon not only to the resources in your executable file, but also as the .ICON extended attribute. File Manager will display this icon next to the name of the executable file.

Example

This example defines an icon whose icon identifier is 11. The icon resource is copied from the file custom.ico.

```
ICON 11 custom.ico
```

ICON Statement (Control)

Syntax:

```
  ICON icon-id, id, x, y, width, height, [style]
```

This form of the ICON statement creates an icon control. This control is an icon displayed in a dialog box. The ICON statement, which you can use only in a DIALOG or WINDOW statement, defines the icon-resource identifier, icon-control identifier, position, and attributes of a control window. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_ICON. For the ICON statement, the width and height fields are ignored; the icon automatically sizes itself.

icon-id	Specifies the resource identifier of an icon that is defined elsewhere in the resource file.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies a reserved value. Can be set to zero.
height	Specifies a reserved value. Can be set to zero.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates an icon control whose icon identifier is 99.

```
ICON 99, 101, 10, 10, 0, 0
```

---

if Directive

Syntax:

```
if constant-expression
```

The if directive controls conditional compilation of the resource file by checking the specified constant expression. If the constant expression is nonzero, if directs the compiler to continue processing statements up to the next endif, else, or elif directive and then skip to the statement after the endif directive. If the constant expression is zero, if directs the compiler to skip to the next endif, else, or elif directive.

constant-expression	Specifies the expression to be checked. This value is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example

This example compiles the BITMAP statement only if the value assigned to the name "Version" is less than 3.

```
#if Version < 3
BITMAP 1 errbox.bmp
#endif
```

---

ifdef Directive

Syntax:

```
ifdef name
```

The ifdef directive controls conditional compilation of the resource file by checking the specified name. If the name has been defined by using a define directive or by using the -d command-line option of rc, ifdef directs the compiler to continue with the statement immediately after the ifdef directive. If the name has not been defined, ifdef directs the compiler to skip all statements up to the next endif directive.

name	Specifies the name to be checked by the directive.

Example

This example compiles the BITMAP statement only if the name "Debug" is defined.

```
#ifdef Debug
BITMAP 1 errbox.bmp
#endif
```

---

ifndef Directive

Syntax:

```
ifndef name
```

The ifndef directive controls conditional compilation of the resource file by checking the specified name. If the name has not been defined or if its definition has been removed by using the undef directive, ifndef directs the compiler to continue processing statements up to the next endif, else, or elif directive and then skip to the statement after the endif directive. If the name is defined, ifndef directs the compiler to skip to the next endif, else, or elif directive.

name	Specifies the name to be checked by the directive.

Example

This example compiles the BITMAP statement only if the name "Optimize" is not defined.

```
#ifndef Optimize
BITMAP 1 errbox.bmp
#endif
```

---

include Directive

Syntax:

```
include filename
```

The include directive causes RC to process the file specified in the filename field. This file should be a header file that defines the constants used in the resource script file. Only the #define directives in the specified file are processed; all other statements are ignored by the Resource Compiler.

filename	Specifies the OS/2 name of the file to be included. This value must be an ASCII string enclosed either in double quotation marks (if the file is in the current directory) or in less-than and greater-than characters (<>) (if the file is in the directory specified by -i command-line options or by the INCLUDE environment variable). You must give a full path enclosed in double quotation marks if the file is not in the current directory or in the directory specified by -i command-line options or by the INCLUDE environment variable.

Comments

The filename field is handled as a C string. Therefore, you must include two backslashes wherever one is required in the path. (As an alternative, you can use a single forward slash (/) instead of two backslashes.)

Example

This example processes the header files OS2.H and HEADERS\MYDEFS.H\I while compiling the resource script file.

```
#include <os2.h>
#include "headers\\\\mydefs.h"
```

LISTBOX Statement

Syntax:

```
LISTBOX id, x, y, width, height[, style]
```

The LISTBOX statement creates commonly used controls for a dialog box or window. The control is a rectangle containing a list of user-selectable strings, such as file names.

The LISTBOX statement, which you can use only in a DIALOG or WINDOW statement, defines the identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_LISTBOX. If you do not specify a style, the default style is WS_TABSTOP.

id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression

	consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_LISTBOX. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a list-box control whose identifier is 101.

```
LISTBOX 101, 10, 10, 100, 100
```

LTEXT Statement

Syntax:

```
  LTEXT text, id, x, y, width, height [, style]
```

The LTEXT statement creates a left-aligned text control. The control is a simple rectangle displaying the given text left-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The LTEXT statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of the control. The predefined class for this control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, DT_LEFT, and WS_GROUP.

text	Specifies text that is left-aligned in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a left-aligned text control that is labeled "Filename."

```
LTEXT "Filename", 101, 10, 10, 100, 100
```

MENU Statement

Syntax:

```
  MENU menu-id  [load-option] [mem-option] [codepage]
  BEGIN
  menuitem-definition
      .
      .
      .
```

```
END
```

The MENU statement defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu is a special input tool that lets a user choose commands from a list of command names. A menu resource can be loaded from the executable file when needed by using the WinLoadMenu function.

You can provide any number of MENU statements in a resource script file, but each statement must specify a unique menu-id value. You can provide any number of menuitem-definition statements in the menu. These define the submenus and menu items (commands) in the menu. The order of the statements defines the order of the menu items.

menu-id	Specifies the menu-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following: PRELOAD — System loads the resource when the application starts. LOADONCALL — System loads the resource when the application calls the WinLoadMenu function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following: FIXED — System keeps the resource at a fixed memory location. MOVEABLE — System moves the resource as necessary to compact memory. DISCARDABLE — System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage	Specifies a codepage value. For a list of valid code pages see CODEPAGE Statement.
menuitem-definition	Specifies a PRESPARAMS, MENUITEM, or SUBMENU statement. You can use one or more PRESPARAMS statements to control the appearance of a menu, such as the font and the foreground and background colors. If used, PRESPARAMS statements must be the first statements following the BEGIN keyword. For details about the PRESPARAMS statement, see PRESPARAMS Statement. MENUITEM and SUBMENU statements define the individual commands or submenus in the given menu. For details about these statements, see MENUITEM Statement and SUBMENU Statement.

Example

This example creates a menu resource whose menu identifier is 1. The menu contains a menu item named Alpha and a submenu named Beta. The submenu contains two menu items, Item 1 and Item 2.

```
MENU 1
BEGIN
    MENUITEM "Alpha", 100
    SUBMENU "Beta", 101
    BEGIN
        MENUITEM "Item 1", 200
        MENUITEM "Item 2", 201, , MIA_CHECKED
    END
END
```

MENUITEM Statement

Syntax:

```
MENUITEM text, menu-id[,  menuitem-style] [, menuitem-attribute]
```

The MENUITEM statement creates a menu item for a menu. The statement, permitted only in a MENU or SUBMENU statement, defines the text, identifier, and attributes of a menu item. The system displays the text when it displays the corresponding menu. If the user chooses the menu item, the system generates a WM_COMMAND message that includes the specified menu-item identifier and sends it to the window

owning the menu.

MENUITEM SEPARATOR

The alternative form of the MENUITEM statement, MENUITEM SEPARATOR, creates a menu separator. A menu separator is a horizontal dividing bar between two menu items in a submenu. The separator is not active - that is, the user cannot choose it, it has no text associated with it, and it has no identifier.

text

Specifies the text of the menu item. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must include the double quotation mark twice. The tilde character (~) and the \t and \a character combinations have special meanings in the string; for details, see the "Comments" section.

If the menuitem-style field is MIS_BITMAP, item-name must be a bit-map identifier instead of a name. The bit-map identifier must have been previously defined using a BITMAP statement, must be preceded by the \b character, and must be enclosed in double quotation marks.

menu-id

Specifies the menu-item identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.

Duplicate menu-item identifiers are allowed, but menu items with non-unique identifiers cannot receive messages.

Character strings cannot be used as resource identifiers for this statement.

menuitem-style

Specifies the menu-item style. This value can be a combination of the following:

MIS_BITMAP	Specifies that item-name is a bit map identifier.
MIS_BREAK	Specifies that the menu has multiple columns of items in one pull-down menu or multiple lines of menus in the top-level menu.
MIS_BREAKSEPARATOR	Specifies that the menu has a vertical line between the columns in a pull-down menu.
MIS_BUTTONSEPARATOR	Specifies that the user can activate the menu item only by using the mouse. The text is centered in the item, rather than left justified. This option is used for the Help item on the right side of the menu bar.
MIS_HELP	Specifies that the menu item generates a WM_HELP message.
MIS_OWNERDRAW	Specifies that the menu item is drawn by the owner window.
MIS_SEPARATOR	Specifies that the menu item is a menu separator.
MIS_STATIC	Specifies that the user cannot choose the menu item.
MIS_SUBMENU	Specifies that the MENUITEM statement is to be treated as a SUBMENU statement. When you specify this option, you must follow the MENUITEM statement with a BEGIN and END clause, as in a SUBMENU statement. You may include a PRESPARAMS statement immediately after the BEGIN keyword.
MIS_SYSCOMMAND	Specifies that the menu item generates a WM_SYSCOMMAND message.
MIS_TEXT	Specifies that item-name is a character string. This is the default option.

menuitem-attribute

Specifies the menu-item attributes. This value can be a combination of the following:

MIA_CHECKED	Places a check mark next to the menu-item name.
MIA_DISABLED	Disables the menu item, preventing the system from generating a message when the user chooses the command.

MIA_FRAMED	Places a frame (heavy border) around the menu item.
MIA_HILITED	Places a highlight on the menu-item name when it is displayed, by inverting the name and background.
MIA_NODISMISS	Causes a submenu or menu item to remain displayed after the user chooses an item.

Comments

You can use the \t or \a character combination in any item name. The \t character inserts a tab when the name is displayed and is typically used to separate the menu-item name from the name of an accelerator key. The \a character aligns to the right all text that follows it. These characters are intended to be used for menu items in submenus only. The width of the displayed submenu is always adjusted so that there is at least one space (and usually more) between any pieces of text separated by a \t or a \a. (When compiling the menu resource, the compiler stores the \t and \a characters as control characters. For example, the \t is stored as 0x09.)

A tilde (~) character in the item name indicates that the following character is used as a mnemonic character for the item. When the menu is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the menu item by pressing the key corresponding to the underlined mnemonic character.

Example

This example creates a menu item named Alpha. The item identifier is 101.

```
MENUITEM "Alpha", 101
```

This example creates a menu item named Beta. The item identifier is 102. The menu item has a text style and a checked attribute.

```
MENUITEM "Beta", 102, MIS_TEXT, MIA_CHECKED
```

This example creates a menu separator between menu items named Gamma and Delta.

```
MENUITEM "Gamma", 103
MENUITEM SEPARATOR
MENUITEM "Delta", 104
```

This example creates a menu item that has a bit map instead of a name. The bit-map identifier, 1, is first defined using a BITMAP statement. The identifier for the menu item is 301. Note that a sign must be placed in front of the bit map identifier in the MENUITEM statement.

```
BITMAP 1 mybitmap.bmp

MENUITEM "#1", 301, MIS_BITMAP
```

MESSAGETABLE Statement

Syntax:

```
  MESSAGETABLE  [load-option] [mem-option] [codepage]
  BEGIN
  string-id string-definition
      .
      .
      .
  END
```

The MESSAGETABLE statement creates one or more string resources for an application. A string resource is a null-terminated character string that has a unique string identifier. A string resource can be loaded from the executable file when needed by using the DosGetResource function with the RT_MESSAGE resource type. RT_MESSAGE resources are bundled together in groups of 16, with any missing IDs replaced with zero length strings. Each group, or bundle, is assigned a unique sequential identifier. The resource string identifier is not necessarily the same as the identifier specified when using DosGetResource. The formula for calculating the identifier of the resource bundle, for use in DosGetResource, is as follows:

```
bundle ID = (id / 16) + 1
```

where id is the string identifier assigned in the RC file.

Thus, bundle 1 contains strings 0 to 15, bundle 2 contains strings 16 to 31, and so on. Once the address of the bundle has been returned by DosGetResource (using the calculated identifier), the buffer can be parsed to locate the particular string within the bundle. The number of the string is calculated by the formula:

```
string = id % 16
```

(string = remainder for id/16).

The buffer returned consists of the CodePage of the strings in the first USHORT, followed by the 16 strings in the bundle. The first BYTE of each string is the length of the string (including the null terminator), followed by the string and the terminator. A zero length string is represented by two bytes: 01 (string length) followed by the null terminator.

You can provide any number of MESSAGETABLE statements in a resource script file. The compiler treats all the strings from the various MESSAGETABLE statements as if they belonged to a single statement. This means that no two strings in a resource script file can have the same string identifier.

Although the MESSAGETABLE and STRINGTABLE statements are nearly identical, most applications use the STRINGTABLE statement instead of the MESSAGETABLE statement to create string resources.

load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following: PRELOAD — System loads the resource when the application starts. LOADONCALL — System loads the resource when the application calls the DosGetResource or DosGetResource2 function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following: FIXED — System keeps the resource at a fixed memory location. MOVEABLE — System moves the resource as necessary to compact memory. DISCARDABLE — System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage	Specifies a code page value. See CODEPAGE Statement for a list of valid code pages.
string-id	Specifies the character-string identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. The value can be specified in decimal or hexadecimal notation. Each string identifier in a resource script file must be unique.
string-definition	Specifies a character string. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must provide the double quotation mark twice.

Comments

You can continue a string on multiple lines by terminating the line with a backslash (\) or by terminating the line with a double quotation mark (") and then starting the next line with a double quotation mark.

Example

This example creates two string resources whose string identifiers are 1 and 2.

```
MESSAGETABLE
BEGIN
    1 "Filename not found"
    2 "Cannot open file for reading"
END
```

MLE Statement

Syntax:

```
  MLE text, id, x, y, width, height[, style]
```

The MLE statement creates a multiple-line entry-field control. The control is a rectangle in which the user can type and edit multiple lines of text. The control displays a pointer when the user selects it. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the BACKSPACE and DELETE keys. By using the mouse or the DIRECTION keys, the user can select the character or characters to delete or select the place to insert new characters. The MLE statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_MLE. If you do not specify a style, the default style is MLS_BORDER, WS_GROUP, and WS_TABSTOP.

text	Specifies text that is displayed in the rectangular area of the control. If the MLS_READONLY style is not specified, the user can edit the text. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_MLE. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a multiple-line entry-field control that is not labeled.

```
MLE "", 101, 10, 10, 50, 100
```

NOTEBOOK Statement

Syntax:

```
  NOTEBOOK   id, x, y, width, height[, style]
```

The NOTEBOOK statement creates a notebook control within the dialog window. This control is used to organize information on individual pages so that it can be located and displayed easily. The NOTEBOOK statement defines the identifier, position, dimensions, and attributes of a notebook control. The predefined class for this control is WC_NOTEBOOK. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE.

id	Specifies the control identifier. The value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_NOTEBOOK. You can use the bitwise OR (|) operator to combine styles.

Comments

The NOTEBOOK statement is used only in a DIALOG or WINDOW statement.

Example

This example creates a notebook control at position (20, 20) within the dialog window. The notebook has a width of 200 character units and a height of 50 character units. Its resource identifier is 201. The tabs style BKS_ROUNDEDTABS specification overrides the notebook default style of square tabs. The default styles WS_TABSTOP and WS_GROUP are both in effect, though only the latter is specified.

```
#define    IDC_NOTEBOOK     201
#define    IDD_NOTEBOOKDLG  503
DIALOG "Notebook", IDD_NOTEBOOKDLG, 11, 11, 420, 420, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    NOTEBOOK   IDC_NOTEBOOK, 20, 20, 200, 400, BKS_ROUNDEDTABS | WS_GROUP
  END
```

POINTER Statement

Syntax:

```
  POINTER pointer-id  [load-option]  [ mem-option] [codepage] filename
```

The POINTER statement defines a pointer resource for an application. A pointer resource, typically created by using the OS/2 Icon Editor, is a bit map defining the shape of the mouse pointer on the screen. The POINTER statement copies the pointer resource from the file specified in the filename field and adds it to the application's other resources. A pointer resource can be loaded from the executable file when needed by using the WinLoadPointer function.

You can provide any number of POINTER statements in a resource script file, but each statement must specify a unique pointer-id value.

pointer-id	Specifies the pointer-resource identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.	
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:	
	PRELOAD	System loads the resource when the application starts.
	LOADONCALL	System loads the resource when the application calls the WinLoadPointer function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:	
	FIXED	System keeps the resource at a fixed memory location.
	MOVEABLE	System moves the resource as necessary to compact memory.
	DISCARDABLE	System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage	Specifies a code page value. See CODEPAGE Statement for a list of valid code pages.	
filename	Specifies the name of the file containing the pointer resource. If the file is not in the current directory, filename must be preceded by a full path.	

Example

This example defines a pointer whose pointer identifier is 10. The pointer resource is copied from the file custom.cur.

```
POINTER 10 custom.cur
```

PRESPARAMS Statement

Syntax:

```
PRESPARAMS presparam, value, presparam, value, ...
```

The PRESPARAMS statement defines presentation fields that customize a dialog box, menu, window, or control. PRESPARAMS data is a series of types and values. The window procedure of the dialog box, menu, window or control receives and processes this data when the item is created. The data for custom controls can be in any format.

presparam	Specifies a presentation-field type.
value	Specifies the presentation-field value.

Comments

PRESPARAMS is often used to supply data to control the appearance of the customized window when it is first created. For example, the PRESPARAMS statement may specify the colors to be used in the window.

Example

This example creates a menu resource with a menu identifier of 1. The PRESPARAMS statement specifies that the following three menu items be displayed in the 12-point Helvetica font.

```
MENU 1
BEGIN
    PRESPARAMS PP_FONTNAMESIZE, "12.Helv"
    MENUITEM "New", 100
    MENUITEM "Open", 101
    MENUITEM "Save", 102
END
```

PUSHBUTTON Statement

Syntax:

```
PUSHBUTTON text, id, x, y, width, height[, style]
```

The PUSHBUTTON statement creates a pushbutton control. The control is a round-cornered rectangle containing the given text. The control sends a message to its parent whenever the user chooses the control. The PUSHBUTTON statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_PUSHBUTTON and WS_TABSTOP.

text	Specifies text that is centered in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a pushbutton control that is labeled "OK."

```
PUSHBUTTON "OK", 101, 10, 10, 100, 100
```

RADIOBUTTON Statement

Syntax:

```
  RADIOBUTTON text, id, x, y, width, height[, style]
```

The RADIOBUTTON statement creates a radio-button control. The control is a small circle that has the given text displayed to its right. The control highlights the circle and sends a message to its parent window when the user selects the button. The control removes the highlight and sends a message when the button is next selected. The RADIOBUTTON statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of a control window. The predefined class for this control is WC_BUTTON. If you do not specify a style, the default style is BS_RADIOBUTTON.

text	Specifies text that is displayed to the right of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice. A tilde (~) character in the text indicates that the following character is used as a mnemonic character for the control. When the control is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the control by pressing the key corresponding to the underlined mnemonic character.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_BUTTON. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a radio-button control that is labeled "Italic."

```
RADIOBUTTON "Italic", 101, 10, 10, 24, 50
```

RCDATA Statement

Syntax:

```
  RCDATA resource-id
  BEGIN
  data-definition, data-definition   ...
       .
       .
       .
```

```
END
```

The RCDATA statement defines a custom-data resource for an application. The custom data can be in whatever format the application requires. You can provide any number of RCDATA statements in a resource script file, but each statement must specify a unique resource-id value. A custom-data resource can be loaded from the executable file when needed by using the DosGetResource or DosGetResource2 functions with the RT_RCDATA resource type.

resource-id	Specifies the custom-data identifier. This value must be an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.
data-definition	Specifies the custom data. The data may be simple expressions or strings.

Example

This example defines custom data that has a resource identifier of 5.

```
RCDATA 5
BEGIN
    "E. A. Poe", 1849, -32, 3L, 0x8000000l, 3+4+5
END
```

RCINCLUDE Statement

Syntax:

```
RCINCLUDE filename
```

The RCINCLUDE statement causes RC to process the resource script file specified in the filename field along with the current resource script file. The contents of both files are compiled by RC and the results are placed in one binary resource file and/or executable file.

filename	Specifies the name of the resource script file to be included. If the file is not in the current directory, filename must be preceded by a full path.

Comments

RCINCLUDE statements are processed before any other processing is done.

When specifying a high performance file system (HPFS) file name on an RCINCLUDE statement, enclose the path and file name in double quotes; for example:

```
RCINCLUDE "d:\project\long dialog.dlg"
```

Double quotes enables the Resource Compiler to recognize a name containing embedded blank characters.

Example

This example includes the file DIALOGS.RC as part of the current resource script file.

```
RCINCLUDE dialogs.rc
```

RESOURCE Statement

Syntax:

```
RESOURCE type-id resource-id [load-option] [mem-option]
         [code-page] filename
```

```
or

RESOURCE type-id resource-id [load-option] [mem-option]
         [code-page]
BEGIN
data-definition [, data-definition]...
...
END
```

The RESOURCE statement defines a custom resource for an application. A custom resource can be any data in any format. The RESOURCE statement copies the custom resource from the specified file or inline data, and adds it to the application's other resources. A custom resource can be loaded from the executable file when needed by using the DosGetResource or DosGetResource2 function and specifying the resource's type and resource identifier.

The custom resource data can be defined in a separate file or as inline data in the input script. This is reflected in the two formats that can be used for this statement. The first format is used when the custom resource data is being read from a file. The second format is used when the data consists of a block of raw source data that is defined inline in the input script.

You can provide any number of RESOURCE statements in a resource script file, but each statement must specify a unique combination of type-id and resource-id values. That is, RESOURCE statements having the same type-id value are permitted as long as the resource-id value for each is unique.

type-id	Specifies the custom-resource type. This value must be an integer in the range 256 through 65535, or a simple expression that evaluates to a value in that range. (Values 0 through 255 are reserved.)
resource-id	Specifies the custom-resource identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following: PRELOAD — System loads the resource when the application starts. LOADONCALL — System loads the resource when the application calls the DosGetResource or DosGetResource2 function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following: FIXED — System keeps the resource at a fixed memory location. MOVEABLE — System moves the resource as necessary to compact memory. DISCARDABLE — System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
codepage	Specifies a code page value. See CODEPAGE Statement. for a list of valid code pages.
filename	Specifies the name of the file containing the custom resource. If the file is not in the current directory, filename must be preceded by a full path.
data-definition	Specifies a custom data definition. The data can be a simple expression or a string. Integers can be specified in decimal, octal, or hexadecimal format. Data definitions in series on the same line are separated by commas. An integer specified without the suffix L must be in the range -32768 through 65535. An integer with an L suffix must be within the range -2147483648 through 4294967295. String data is specified within quotes. **Note:** The Resource Compiler does not append a null character to the end of these strings as it does for RCDATA blocks; any required null characters must be written as \0 within the data string.

Example

This example defines a custom resource whose type identifier is 300 and whose resource identifier is 14. The custom resource is copied from the file CUSTOM.RES.

```
RESOURCE 300 14 custom.res
```

RTEXT Statement

Syntax:

```
RTEXT text, id, x, y, width, height[, style]
```

The RTEXT statement creates a right-aligned text control. The control is a simple rectangle displaying the given text right-aligned in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line. The RTEXT statement, which you can use only in a DIALOG or WINDOW statement, defines the text, identifier, dimensions, and attributes of the control. The predefined class for the control is WC_STATIC. If you do not specify a style, the default style is SS_TEXT, DT_RIGHT, and WS_GROUP.

text	Specifies text that is right-aligned in the rectangular area of the control. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the text, you must include the double quotation mark twice.
id	Specifies the control identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
y	Specifies the y-coordinate of the lower-left corner of the control. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog box, window, or control containing the specified control.
width	Specifies the width of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. This value can be a combination of the styles specified for WC_STATIC. You can use the bitwise OR (|) operator to combine styles.

Example

This example creates a right-aligned text control that is labeled "Filename."

```
RTEXT "Filename", 101, 10, 10, 100, 100
```

SLIDER Statement

Syntax:

```
SLIDER   id, x, y, width, height[, style]
```

The SLIDER statement creates a slider control within the dialog window. This control lets the user set, display, or modify a value by moving a slider arm along a slider shaft. The SLIDER statement defines the identifier, position, dimensions, and attributes of a slider control. The predefined class for this control is WC_SLIDER. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE.

id	Specifies the control identifier. The value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. The value can be any combination of the styles specified for WC_SLIDER. You can use the bitwise OR (|) operator to combine styles.

Comments

The SLIDER statement is only used in a DIALOG or WINDOW statement.

Example

This example creates a slider control at position (40, 30) within the dialog window. The slider has a width of 120 character units and a height of 2 character units. Its resource identifier is 101. The style specification SLS_BUTTONSLEFT adds buttons to the left of the slider shaft. The default styles WS_TABSTOP and WS_VISIBLE are both in effect, though only the latter is specified.

```
#define    IDC_SLIDER       101
#define    IDD_SLIDERDLG    502
DIALOG "Slider", IDD_SLIDERDLG, 11, 11, 200, 240, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    SLIDER   IDC_SLIDER, 40, 30, 120, 16, SLS_BUTTONSLEFT | WS_VISIBLE
  END
```

SPINBUTTON Statement

Syntax:

```
  SPINBUTTON   id, x, y, width, height[, style]
```

The SPINBUTTON statement creates a spin button control within the dialog window. This control gives the user quick access to a finite set of data. The SPINBUTTON statement defines the identifier, position, dimensions, and attributes of a spin button control. The predefined class for this control is WC_SPINBUTTON. If you do not specify a style, the default style is WS_TABSTOP, WS_VISIBLE, and SPBS_MASTER.

id	Specifies the control identifier. The value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. The value is any combination of the styles specified for WC_SPINBUTTON. You can use the bitwise OR (|) operator to combine styles.

Comments

The SPINBUTTON statement is used only in a DIALOG or WINDOW statement.

Example

This example creates a spin-button control at position (80, 20) within the dialog window. The spin button has a width of 60 character units and a height of 3 character units. Its resource identifier is 302. The style specification SPBS_NUMERICONLY creates a control which accepts only the digits 0-9 and virtual keys. The default styles SPBS_MASTER, WS_TABSTOP, and WS_VISIBLE are all in effect, though only WS_TABSTOP is specified.

```
#define    IDC_SPINBUTTON   302
#define    IDD_SPINDLG    502
DIALOG "Spin button", IDD_SPINDLG, 11, 11, 200, 240, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    SPINBUTTON  IDC_SPINBUTTON, 80, 20, 60, 24, SPBS_NUMERICONLY | WS_TABSTOP
  END
```

STRINGTABLE Statement

Syntax:

```
STRINGTABLE  [load-option] [mem-option] [codepage]
BEGIN
string-id string-definition
    .
    .
    .
END
```

The STRINGTABLE statement creates one or more string resources for an application. A string resource is a null-terminated character string that has a unique string identifier. A string resource can be loaded from the executable file when needed by using the WinLoadString or with DosGetResource with the RT_STRING resource type. RT_STRING resources are bundled together in groups of 16, with any missing IDs replaced with zero length strings. Each group, or bundle, is assigned a unique sequential identifier. The resource string identifier is not necessarily the same as the identifier specified when using DosGetResource. The formula for calculating the identifier of the resource bundle, for use in DosGetResource, is as follows:

```
bundle ID = (id / 16) +1
```

where id is the string ID assigned in the RC file.

Thus, bundle 1 contains strings 0 to 15, bundle 2 contains strings 16 to 31, and so on. Once the address of the bundle has been returned by DosGetResource (using the calculated identifier), the buffer can be parsed to locate the particular string within the bundle. The number of the string is calculated by the formula:

```
string = id % 16
```

(string = remainder for id/16).

The buffer returned consists of the CodePage of the strings in the first USHORT, followed by the 16 strings in the bundle. The first BYTE of each string is the length of the string (including the null terminator), followed by the string and the terminator. A zero length string is represented by two bytes: 01 (string length) followed by the null terminator.

You can provide any number of STRINGTABLE statements in a resource script file. The compiler treats all the strings from the various STRINGTABLE statements as if they belonged to a single statement. This means that no two strings in a resource script file can have the same string identifier.

load-option — Specifies when the system loads the resource from the executable file into memory. This value must be one of the following:

PRELOAD	System loads the resource when the application starts.
LOADONCALL	System loads the resource when the application calls the WinLoadString function. This is the default option.

mem-option — Specifies how the system manages the resource when it is in memory. This value must be one or more of the following:

FIXED	System keeps the resource at a fixed memory location.
MOVEABLE	System moves the resource as necessary to compact memory.
DISCARDABLE	System discards the resource if it is no longer needed.

The default setting is MOVEABLE and DISCARDABLE.

code-page — Specifies a code page value. See CODEPAGE Statement for a list of valid code page values.

string-id — Specifies the character-string identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges. The value can be specified in decimal or hexadecimal notation. Each string identifier in a resource script file must be unique.

string-definition — Specifies a character string. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must include the double quotation mark twice.

Comments

You can continue a string on multiple lines by terminating the line with a backslash (\) or by terminating the line with a double quotation mark (") and then starting the next line with a double quotation mark.

Example

This example creates two string resources whose string identifiers are 1 and 2.

```
#define IDS_HELLO    1
#define IDS_GOODBYE  2

STRINGTABLE
BEGIN
    IDS_HELLO   "Hello"
    IDS_GOODBYE "Goodbye"
END
```

SUBITEMSIZE Statement

Syntax:

```
  SUBITEMSIZE  size
```

The SUBITEMSIZE statement specifies the size, in words, of each help subitem in a help subtable. The minimum size is two words, and each help subitem in a help subtable must be the same size. When used, the SUBITEMSIZE statement must appear after the HELPSUBTABLE statement and before the BEGIN keyword.

You do not need to use the SUBITEMSIZE statement if the help subitems are the default size (2).

size Specifies the size of each help subitem. This value must be an integer.

Example

The SUBITEMSIZE statement in this example specifies that each HELPSUBITEM statement contains three words.

```
HELPSUBTABLE 1
SUBITEMSIZE 3
BEGIN
    HELPSUBITEM IDCLD_FILEMENU, IDHP_FILEMENU, 5
    HELPSUBITEM IDCLD_HELPMENU, IDHP_HELPMENU, 6
END
```

SUBMENU Statement

Syntax:

```
  SUBMENU text, submenu-id [, menuitem-style]
  BEGIN
  menuitem-definition
      .
      .
      .
  END
```

The SUBMENU statement creates a submenu for a given menu. A submenu is a vertical list of menu items from which the user can choose a command.

You can provide any number of SUBMENU statements in a MENU statement, but each SUBMENU statement must specify a unique

submenu-id value. You can provide any number of menuitem-definition statements in the SUBMENU statement. These define the menu items (commands) in the menu. The order of the statements determines the order of the menu items.

text	Specifies the text of the submenu. This field must contain zero or more characters enclosed in double quotation marks. Character values must be in the range 1 through 255. If a double quotation mark is required in the string, you must include the double quotation mark twice. A tilde (~) character in the item name indicates that the following character is used as a mnemonic character for the item. When the menu is displayed, the tilde is not shown, but the mnemonic character is underlined. The user can choose the menu item by pressing the key corresponding to the underlined mnemonic character.
submenu-id	Specifies the submenu identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
menuitem-style	Specifies the submenu style. This value can be a combination of MIS_ values. For details on the MIS_ values, see MENUITEM Statement.
menuitem-definition	Specifies a PRESPARAMS or MENUITEM statement. You can use the PRESPARAMS statement to control the appearance of a submenu, such as the font and the foreground and background colors. If used, the PRESPARAMS statement must immediately follow the BEGIN keyword. For details about the PRESPARAMS statement, see PRESPARAMS Statement. The MENUITEM statement defines an individual command in the given menu. For details, see MENUITEM Statement.

Example

This example creates a submenu named Elements. Its identifier is 2. The submenu contains three menu items, which are created by using MENUITEM statements.

```
SUBMENU "Elements", 2
BEGIN
    MENUITEM "Oxygen", 200
    MENUITEM "Carbon", 201, , MIA_CHECKED
    MENUITEM "Hydrogen", 202
END
```

---

undef Directive

Syntax:

```
  undef name
```

This directive removes the current definition of the specified name. All subsequent occurrences of the name are processed without replacement.

name	Specifies the name to be removed. This value is any combination of letters, digits, and punctuation.

Example

This example removes the definitions for the names "nonzero" and "USERCLASS".

```
#undef     nonzero
#undef     USERCLASS
```

VALUESET Statement

Syntax:

```
  VALUESET   id, x, y, width, height[, style]
```

The VALUESET statement creates a value set control within the dialog window. This control lets a user select one choice from a group of mutually exclusive choices. The VALUESET statement defines the identifier, position, dimensions, and attributes of a value set control. The predefined class for this control is WC_VALUESET. If you do not specify a style, the default style is WS_TABSTOP and WS_VISIBLE.

id	Specifies the control identifier. The value is a signed integer -32768 through 32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
y	Specifies the y-coordinate of the lower-left corner of the control. The value is a signed integer -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The coordinate is assumed to be in dialog units and is relative to the origin of the dialog window.
width	Specifies the width of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The width is in n-character units.
height	Specifies the height of the control. The value is any integer 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The height is in 1/8-character units.
style	Specifies the control styles. The value is any combination of the styles specified for WC_VALUESET. You can use the bitwise OR (|) operator to combine styles.

Comments

The VALUESET statement is used only in a DIALOG or WINDOW statement.

Example

This example creates a value set control at position (40, 40) within the dialog window. The value set control has a width of 220 character and a height of 20 character units. Its resource identifier is 302. The style specification VS_ICON creates a control to show items in icon form. The default styles WS_TABSTOP and WS_VISIBLE are both in effect, though only WS_TABSTOP is specified.

```
#define    IDC_VALUESET     302
#define    IDD_VALUESETDLG  501
DIALOG "Value set", IDD_VALUESETDLG, 11, 11, 260, 240, FS_NOBYTEALIGN |
        WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
  BEGIN
    VALUESET  IDC_VALUESET, 40, 40, 220, 160, VS_ICON | WS_TABSTOP
  END
```

WINDOW Statement

Syntax:

```
  WINDOW text, id, x, y, width, height, class[, style[, framectl]]
         data-definitions
[ BEGIN
control-definition
    .
    .
    .
END ]
```

The WINDOW statement creates a window of the specified class. The statement defines the position and dimensions of the window relative to its parent window, as well as the window-box style. The WINDOW statement is typically used in a WINDOWTEMPLATE or FRAME statement.

Typically, only one WINDOW statement is used in a FRAME statement. It defines the client window belonging to the corresponding frame window. The optional BEGIN and END keywords enclose any CONTROL statements that are given with the window. CONTROL statements given in this manner represent child windows belonging to the window created by the WINDOW statement.

text	Specifies the window title if the style specifies a title bar. This field must contain zero or more characters enclosed in double quotation marks. The character values must be in the range 1 through 255. If a double quotation mark is required in the title, you must include the double quotation mark twice.
id	Specifies the window identifier. This value must be a signed integer in the range -32768 through

	32767, an unsigned integer in the range of 1 through 65535, or a simple expression that evaluates to a value in these ranges.
x	Specifies the x-coordinate of the lower-left corner of the window. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in dialog units. The position is relative to the origin of the parent window.
y	Specifies the y-coordinate of the lower-left corner of the window. This value must be a signed integer in the range -32768 through 32767 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in dialog units. The position is relative to the origin of the parent window.
width	Specifies the width of the window. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in n-character units.
height	Specifies the height of the window. This value must be an integer in the range 0 through 65535 or an expression consisting of integers and the addition (+) or subtraction (-) operator. The value is in 1/8-character units.
class	Specifies the window class. This value can be one of the control classes specified in the "Control Classes" table in the *Presentation Manager Programmer Reference* or the name of the window class, enclosed in double quotation marks.
style	Specifies the window style. This value can be any of the window, dialog box, or frame styles specified.
framectl	Specifies the style of the frame controls belonging to the window. This value can be a combination of the styles specified in the table, "Frame-Control Styles." You can use the bitwise OR (|) operator to combine styles.
data-definitions	Specifies a CTLDATA and/or PRESPARAMS statement. These statements define control and presentation data for the window. For more information, see CTLDATA Statement and PRESPARAMS Statement.
control-definition	Specifies a CONTROL statement or any one of several predefined control statements. These statements define the style, position, and dimensions of controls in the window.

Comments

The WINDOW statement can actually contain any combination of CONTROL, DIALOG, and WINDOW statements. Typically, a WINDOW statement contains one or no such statements.

Example

This example creates a client window belonging to the frame window. The client window belongs to the "MyClientClass" window class and has the standard window identifier FID_CLIENT.

```
WINDOWTEMPLATE 1
BEGIN
    FRAME "My Window", 1, 10, 10, 320, 130,
             0, FCF_STANDARD | FCF_VERTSCROLL
    BEGIN
        WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass"
    END
END
```

WINDOWTEMPLATE Statement

Syntax:

```
WINDOWTEMPLATE window-id [load-option] [mem-option] [code-page]
BEGIN
window-definition
    .
    .
    .
END
```

The WINDOWTEMPLATE statement creates a window template. A window template consists of a series of statements that define the window identifier, load and memory options, window dimensions, and controls in the window. The window template can be loaded from the executable file by using the WinLoadDlg function.

You can provide any number of window templates in a resource script file, but each template must have a unique window-id value.

window-id	Specifies the window identifier. This value must be a signed integer in the range -32768 through 32767, an unsigned integer in the range of 1 through 65535, a simple expression that evaluates to a value in these ranges, or a character string.
load-option	Specifies when the system loads the resource from the executable file into memory. This value must be one of the following: PRELOAD — System loads the resource when the application starts. LOADONCALL — System loads the resource when the application calls the WinLoadDlg function. This is the default option.
mem-option	Specifies how the system manages the resource when it is in memory. This value must be one or more of the following: FIXED — System keeps the resource at a fixed memory location. MOVEABLE — System moves the resource as necessary to compact memory. DISCARDABLE — System discards the resource if it is no longer needed. The default setting is MOVEABLE and DISCARDABLE.
code-page	Specifies a code page value. See CODEPAGE Statement for a list of valid code pages.
window-definition	Specifies a WINDOW statement. The statement defines the dimensions and style of the given window. For details about the statement, see WINDOW Statement.

Comments

A WINDOWTEMPLATE statement can contain DIALOG, CONTROL, and WINDOW statements. Typically, only one WINDOW statement is used in the WINDOWTEMPLATE statement.

Dynamic Trace Customizer (TRCUST)

OS/2 provides a mechanism by which developers may dynamically apply tracepoints in their module at run time. This method eliminates all overhead of tracing when tracing is disabled. It also allows the developer to add tracepoints without modifying source code. This reduces the possibility that adding a tracepoint will induce errors into one's code. OS/2 needs a binary file, for each module being dynamically traced, which defines the tracepoints for the module.

There are certain restrictions on the use of dynamic trace which should be noted. These are:

- Executable (.EXE) programs running in processes other than the trace command cannot have dynamic tracepoints applied.
- A dynamic tracepoint cannot be applied to a module running under the Kernel Debugger that has a Breakpoint in place at the same location as the trace point.
- The trace command is unable to apply dynamic tracepoints to any module for whom DosLoadModule is not applicable. This precludes the following:
    - Physical Device Drivers
    - Virtual Device Drivers
    - File System Device Drivers

  The DTRACE utility (shipped with the IBM Developer Connection and OS/2 Warp 4) provides a means of overcoming this particular restriction.

The Trace Customizer (TRCUST) converts tracepoint definitions from a trace source file (TSF) into dynamic tracepoints for the trace definition file (TDF), and into formatting rules in the trace format file (TFF).

Definitions

.TSF	An ASCII file created by a developer which defines all dynamic tracepoints for a given module. TRCUST currently allows at most only one major code per TSF.
.TDF	A binary file, created by TRCUST, using the **.TSF** file as input. This file defines all tracepoints in the module in a manner acceptable to OS/2. This is used by the Trace Utility, TRACE.
.TFF	A binary file also created by TRCUST using the **.TSF** file. This file defines how all tracepoints will be formatted. This is used by the Trace Formatter, TRACEFMT.

major code	A byte value used to identify the module being traced. TRCUST allows at most only one major code per TSF.
minor code	A word value used to uniquely identify each tracepoint.
GROUP	A value used to identify this tracepoint with tracepoints of the same category. Examples are MEM for memory management and PM for Presentation Manager. For an example of uses of groups, see the online help for the **TRACE** command.
TYPE	A value used to associate a subset of dynamic trace events within a module. Examples are PRE for pre-invocation, POST for post-invocation and API for API calls within a module. For an explanation and examples of uses of types, see the online help for the **trace** command.

File Naming and Location

The TDF file name is the same as the module to be traced, but has a file extension of TDF. The TFF has a name of the form TRC00xx.TFF where xx is the major code, for example, a module with major code 0xC2 will generate a TFF with the name TRC00C2.TFF. This naming convention is used to allow TRACEFMT to dynamically generate the TFF name given only the major code.

TRCUST can be invoked to process a TSF or to combine several TFF files into a single TFF. For processing a TSF, TRCUST is given the name of a TSF, and optionally:

- the desired name of the resulting TDF
- the MAP file name
- the error message level

TRCUST will store the TSF tracepoint formatting specifications in the TFF file and if the tracepoint specified was not for a static tracepoint, the TSF tracepoint definition will be converted into the format required by the Trace Utility and stored in the TDF file. On errors, TRCUST will display appropriate messages, skip any tracepoint with errors in its definition, and continue processing the next tracepoint definition.

For combining TFF files that use the same major code, TRCUST is given the name of the file containing the TFF filenames to combine and the name of the file to contain the combined trace format statements.

TRCUST will issue an error message and abort processing under the following conditions:

- the TSF cannot be opened
- when combining TFF files, if any TFF input files cannot be read or if all TFF input files do not use the same major code
- when defining dynamic tracepoints, if the executable module to contain the tracepoints cannot be read
- the TDF, or TFF files cannot be written to
- an error in the header definition in the TSF
- a missing ending quote in the TSF

Note: TRCUST always returns 0 so that, when invoking it from a makefile, processing of the rest of the makefile can continue if TRCUST aborts.

Combine TFF files when several modules that use dynamic tracing use the same major code. The Trace Formatter can only use one TFF file per major code to get formatting information from. After the TSF file for each module is run through TRCUST to produce a TDF and TFF file, TRCUST can be invoked again, this time using the combine TFF files option and a file that only contains the paths to all the TFF files using the same major code. TRCUST will read all the TFF files. If all TFF files don't use the same major code, TRCUST will issue an error message and abort. TRCUST will read each trace format record from the TFF files and write them (in ascending order according to minor code) to the destination TFF file given.

Invoking the Trace Customizer

The Trace Customizer is a protect mode only program and must therefore be run under OS/2. TRCUST may be invoked to combine TFF files or to process a TSF. The syntax for combining TFF files is as follows:

```
[d:][path]TRCUST [d:][path]tffsource /C=[d:][path]tffdest [/Wn]
```

where:

TRCUST — is the name of the Trace Customizer program. A drive and path may optionally be specified to explicitly define the location of the Trace Customizer program, otherwise the program is searched for in the current directory, followed by looking along the path defined by the PATH environment variable.

[d:][path]tffsource — is the name of a file containing fully qualified (including extensions) pathnames of TFF files to combine. Each TFF file must use the same major code and each filename in the **tffsource** file is separated by white space. This will combine all TFF files for the same major code into a single TFF file. If duplicate minor code format definitions are found, the first format definition for the minor code remains valid, the duplicates are discarded and a warning message is issued. If no path is provided the **tffsource** file is searched for in the current directory, followed by using the current value of DPATH.

[d:][path]tffdest — is the name of the trace format destination file to store the combined trace format definitions.

/Wn — (optional) is the level of error messages to be displayed, where **n** can be 0, 1, or 2. The possible message levels are shown below along with the messages that each displays:

Level	Messages
0	fatal and severe messages
1	fatal, severe, and error messages
2	all (fatal, severe, error, and warning) messages

A message level of 2 is the default.

An example of a tffsource file for using the combine TFF files option of TRCUST is:

```
\TFF\PROG1\TRC00C2.TFF \TFF\PROG2\TRC00C2.TFF
\TFF\PROG3\TRC00C2.TFF \TFF\PROG4\TRC00C2.TFF
```

To invoke TRCUST to combine TFF files using the above file as input (assume filename is \TFF\PROG\TFF00C2) and output the combined format statements into file \TFF\PROG\TR\TRC00C2.TFF is:

```
TRCUST \TFF\PROG\TFF00C2 /C=\TFF\PROG\TRC00C2.TFF
```

The syntax for processing a TSF file is as follows:

```
[d:][path]TRCUST [d:][path]tsf [[d:][path]tdf] [/M=mapfile] [/Wn]
```

where:

TRCUST — is the name of the Trace Customizer program. A drive and path may optionally be specified to explicitly define the location of the Trace Customizer program, otherwise the program is searched for in the current directory, followed by looking along the path defined by the PATH environment variable.

[d:][path]tsf — is the name of the trace source file. If no file extension is provided then an extension of TSF is assumed. If no path is provided the trace source file is searched for in the current directory, followed by using the current value of DPATH.

[d:][path]tdf — (optional) is the name of the trace definition file to store the dynamic tracepoint definitions. If not specified, the TSF filename is used with an extension of TDF. If no file extension is provided then an extension of TDF is assumed.

/M=mapfile — (optional) defines *mapfile* as the MAP file for this module. The name may be qualified by a drive/directory, otherwise it will be searched for in the current directory followed by the path specified by the DPATH environment variable. If specified as an option, the MAP file must exist and the filename extension must be MAP or TRCUST will abort processing. The mapfile will only be used if a symbol is not found in the symbolic debug information stored in the executable module.

/Wn — (optional) is the level of error messages to be displayed, where **n** can be 0, 1, or 2. The possible message levels are shown below along with the messages that each displays:

Level	Messages

0	fatal and severe messages
1	fatal, severe, and error messages
2	all (fatal, severe, error, and warning) messages

A message level of 2 is the default.

Symbolic Debug Support

Source Level Symbolic Support

If the module has been compiled and linked with the following debug options, then the Trace Customizer can look into the module to extract symbolic information. In this case addresses may be specified symbolically.

- **/Ti** on the C/Set2 language compiler for C source files
- **/CO** on the link command

Note that not all source files must be C language, although only public labels from assembler routines will be found in the symbolic information. You may specify filename and line number, a local variable name or a global variable name when using C routines. All symbolic names are case sensitive when the source was compiled with debug options, but if linking in a C language program that was not compiled with debug options, all symbolic names are case sensitive and begin with an underscore (_) character unless the name is declared with the Pascal calling conventions, in which case the underscore is omitted but the symbolic name is capitalized.

MAP File Support

The Trace Customizer can also use the symbolic information in the MAP file produced by the linker. All public symbols will be listed with their offsets in the module being traced. This is not as complete a support as offered by the debug compile option for C language source files, but it does allow entry points, public labels and global data to be referenced symbolically within the TSF. Note that the use of a MAP file is not language dependent.

Note: When using a MAP file, if the symbolic name is a C language entry point, it will be case sensitive and begin with an underscore (_) character unless it is declared with the Pascal naming convention, in which case the underscore is omitted and the name is capitalized. If the name is not from C language source file, the name is case sensitive.

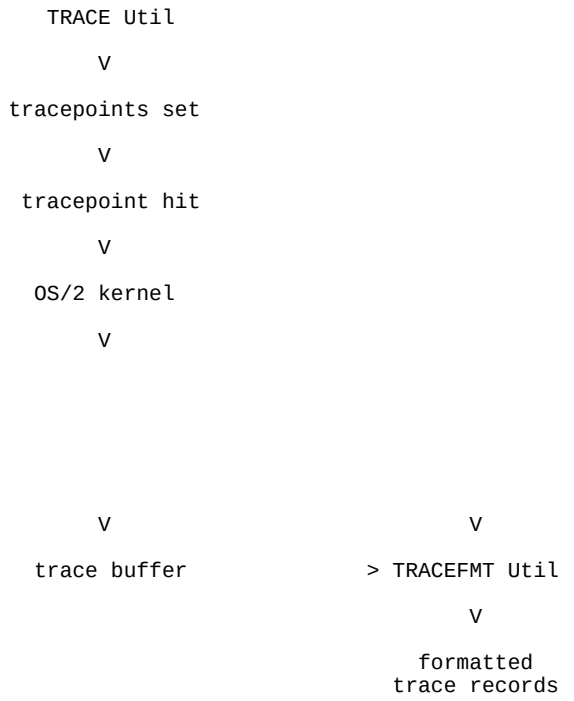
Building a Module

To trace only public procedures, you only need your MAP file that was generated by linking your module.

To trace local variables in C language routines, compile the C programs with the debug option and assemble the ASM routines with public symbols. Link all the OBJs together with the debug option (/CO) and run TRCUST on the executable module. You can now strip the debug information from the executable file by either relinking the OBJs without the debug option or by using a tool to delete the debug information from the executable module file.

TDF and TFF File Usage

The TDF, and TFF files produced by TRCUST are used in the following way:



How TRCUST fits into the system

Tracing a Module

To trace a module do the following:

1. Define the tracepoints and data to be traced in the TSF.
2. Invoke the Trace Customizer using the TSF as input.

   This produces two files, a TDF and a TFF.
3. Put the TDF file in the same directory the module to trace resides, put the TFF file in a directory accessible by TRACEFMT. It is suggested that all TFF files reside in the same subdirectory, an example directory could be \OS2\SYSTEM\TRACE.
4. Invoke the OS/2 TRACE command using the name of the TDF instead of the major code value.

   This activates the tracepoints, causing the trace data to be saved in the system trace buffer.
5. The OS/2 TRACE command can be used to turn tracing off at any time.
6. To display the contents of the trace buffer, invoke the OS/2 TRACEFMT command.

   TRACEFMT uses the major code to determine the TFF file and uses the formatting string corresponding to the minor code value to format the data in the RAS trace buffer and output it to the screen, file or printer.

Symbols and Abbreviations Used in the Document

[...]	denotes optional items.
[... | ... | ...]	denotes a list of optional items, zero or more of which may be chosen.
{... | ... | ...}	denotes a list of items of which ONE must be chosen.
item...	denotes that *item* is repeated zero or more times.
statement,.....	denotes this example is incomplete.
nnn	is a number in the range 0-255 inclusive.
nnnnn	is a number in the range 0-65535 inclusive.

All numbers and values can be entered in decimal form or in C hexadecimal form (0x....).

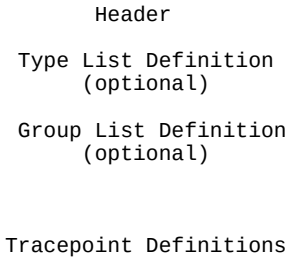
Trace Source File

This section details the statements that can appear within a trace source file.

Examples are given of TRACE statements.

TSF Format

The layout of a trace source file is:



Note:

- Comments may be freely inserted anywhere in the trace source file. A comment is identified by a **;** or by using C syntax comments anywhere in the file. A C comment has start and end delimiters, namely **/*** and ***/**. C type comments may span lines, and may be nested.
- Below are sample TSF files. See Sample Trace Source Files for more examples.

```
;  Sample trace source file depicting dynamic tracing for OS calls compiled
;  with 32-bit addressing

      MODNAME = doscall1.dll
      MAJOR   = 100  /* this is decimal, would be 0x64 if specified hex */
      MAXDATALENGTH = 200 /* max bytes logged per tracepoint is 200 */
```

```
TYPELIST NAME=PRE,ID=1,
         NAME=SYS,ID=0x40,
         NAME=API,ID=128,     /* decimal, if hex would be 0x80 */
         NAME=POST,ID=0x8000

GROUPLIST NAME=MEM,ID=2,
          NAME=FS,ID=0x5,
          NAME=MOU,ID=13,
          NAME=DOS,ID=0x2B    /* would be 43 if decimal */

TRACE   MINOR=0x0001,
        TP=.DosOpen,          /* Pre-invocation tracing on DosOpen */
        TYPE=(PRE,API),
        GROUP=DOS,
        DESC="(OS) DosOpen    Pre-Invocation",
        FMT="Major = %X  Minor = %Y",
        FMT="            EAX = %D",
        FMT="            FileName = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S",
        REGS=(EAX),
        ASCIIZ32=(.FileName,DIRECT,128)

TRACE   MINOR=0x7001,         /* Puts tracept on code at line 28 */
                              /*   of file dosopen1.c.  Debug    */
        TP=@dosopen1.c,28,    /*   info is needed to use this.   */
        TYPE=(API),
        GROUP=DOS,
        DESC="(OS) CheckParm  After Createhandle",
        FMT="                 New handle = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W",
        MEM32=(.handle,DIRECT,2)

TRACE   MINOR=0x8001,         /* Post-invocation tracing at         */
        TP=.DosOpenC,RETEP,   /*   procedure DosOpenC return point. */
        TYPE=(API,POST),      /*   Debug    info is needed to use   */
        GROUP=DOS,            /*   this type of tracepoint.         */
        DESC="(OS) DosOpenC   Post-Invocation",
        FMT="                 Return Code = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W",
        FMT="                 Variable Rec= /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69U",
        MEM32=(.retcode,DIRECT,2),
        /* The following will log a variable length structure.  The */
        /*   second field in the structure is the length of the     */
        /*   record(position var_struct+2).  This LEN parameter must*/
        /*   immediately precede the memory specification defining  */
        /*   the variable length record.                            */
        LEN=(var_struct+2,DIRECT),
        MEM32=(.var_struct,DIRECT,LEN)
```

TSF Header

This defines common information for the module to be traced. The format is:

```
  MODNAME = [d:][path]Name
  MAJOR   = nnn
[ MAXDATALENGTH = nnnn ]
```

where:

d:	is the drive containing the module. If not specified the current drive is used.
path	is the path to the module. If not specified the current path is used.
Name	is the name of the executable module to be traced. If an extension is not specified and the **Name** is not **OS2KRNL**, an extension of DLL is appended to **Name**.
MAJOR=nnn	defines the major trace ID allocated to this module. It may be in the range 1 to 255 decimal or specified 0x1 to 0xFF hex. The default value is 1. The major trace ID is part of the data placed in the trace buffer when a tracepoint is executed.

Only one major code is permitted per module.

Note: OS/2 currently only supports major codes 0x1 - 0x00FF.

MAXDATALENGTH=nnnn (optional) defines the maximum amount of data that a single tracepoint call will insert into the trace buffer.

The length may be in the range 20 to 512 decimal or specified 0x14 to 0x200 hex. The default value is 512. This limit on the amount of data to trace is to avoid yielding the processor when doing dynamic tracing.

Typelist Definition

This defines the optional typelist event IDs. For more description and examples of event types see the online help for the **trace** command.

The format is:

```
TYPELIST NAME=TypeName,ID=TypeValue,
        [NAME=TypeName,ID=TypeValue,]...
```

where:

NAME=Typename
defines a 1-8 byte character string used to reference the TypeValue in the tracepoint definitions. All TypeNames and GroupNames within a TSF must be unique.

ID=TypeValue
defines a bit value of the form 2**y where y in range 0 to 15, permitting a maximum of 16 types to be defined in a single TSF. This can be decimal or specified 0xnnnn for hex.

An example TYPELIST definition follows:

```
TYPELIST NAME=PRE,ID=1,
         NAME=SYS,ID=0x40,
         NAME=API,ID=128,
         NAME=POST,ID=0x8000,.....
```

Grouplist Definition

This defines the optional grouplist IDs. For more description and examples of groups see the online help for the **trace** command.

The format is:

```
GROUPLIST NAME=GroupName,ID=GroupValue,
         [NAME=GroupName,ID=GroupValue,]...
```

where:

NAME=GroupName
defines a 1-8 byte character string used to reference the GroupValue in the tracepoint definitions. There are a maximum of 48 GroupNames allowed in a TSF file. All TypeNames and GroupNames within a TSF must be unique.

ID=GroupValue
defines a word value in the range 1 to 65535 decimal or 0x1 to 0xFFFF hex.

An example GROUPLIST definition follows:

```
GROUPLIST NAME=MEM,ID=2,
          NAME=FS,ID=0x5,
```

```
NAME=MOU,ID=13,.....
```

Tracepoint Definitions

The tracepoint address and the data to be traced are specified by the **TRACE** statement. There are a maximum of 65535 tracepoints permitted in a trace source file.

The format of the TRACE statement is:

```
TRACE   [MINOR=minorcode,]
        TP={@STATIC,|@filename,linenum,|.name[{+|-}offs][,RETEP]},
        [OPCODE=0xnn,]
        [TYPE=(typename[,typename...]),]
        [GROUP=groupnam,]
        [DESC="Tracepoint description",]
        [FMT="Formatting string",]...
        [LEN=(length_spec,flag),]
        [DATA_STMT,]...
```

The TRACE keyword delimits a tracepoint definition statement. The definition is considered complete when the next TRACE keyword is encountered or the end of file is reached. There is one TRACE statement for each tracepoint.

LEN is used to log variable length records. A DATA_STMT must immediately follow the LEN statement. LEN will give the location of a one word field containing the number of bytes to log for the following DATA_STMT.

MINOR Keyword

The MINOR parameter is an optional keyword parameter. If it is specified in the first tracepoint definition, it must be specified in every tracepoint definition. If it is not specified in the first tracepoint definition, it cannot be specified in any of the subsequent tracepoint definitions. It should be coded as:

```
MINOR=nnnnn,
```

where:

nnnnn — is a decimal number from 1 to 65535 or a hex number from 0x1 to 0xFFFF. This represents the minor code for the tracepoint, which must be unique for the major code specified for this module. When tracepoints with duplicate minor codes are encountered, the first is saved and the rest are discarded, and an error message is issued.

If minor codes are not specified in the TSF, TRCUST sequentially provides them, starting with 1, for each tracepoint definition it encounters.

TP Keyword

The TP parameter is a required keyword parameter. If TP is specified more than once for a single tracepoint definition, the tracepoint is discarded. TP has three mutually exclusive definitions which can be coded as:

```
TP=@STATIC,
```

where:

STATIC — defines this tracepoint entry to be used only for creating a trace format statement for the TFF file. No tracepoint definition is created for the TDF, and the only other TRACE parameters that will be used are DESC, MINOR and FMT. This is used to create trace formatting information for static tracepoints. If the TSF contains only @STATIC

directives, no TDF files are created.

```
TP=@filename,linenum,
```

where:

filename — is an ASCII string specifying the name (including extension) of a source filename used in creating the module. The source filename is stored in the debug information contained in the executable module, so debug information must exist to use this parameter. The filename is not case sensitive.

linenum — is a decimal number specifying the line number in the given source file name to place the tracepoint.

Note: Debug information must exist to use this option. The statement at the given source linenum may have been rearranged during compiler optimization, so the developer must use this with caution. If the linenum is not found in the debug information, the tracepoint is applied at the next linenum defined in the debug information and a warning message is issued to the user.

An example to apply a tracepoint to line 35 of file stubfile.c is:

```
TRACE   MINOR=0x700A,              /* puts tracepoint on code at line */
        TP=@stubfile.c,35,.....    /*  35 of source file stubfile.c   */
```

```
TP=.name[{+|-}offs][,RETEP],
```

where:

name
: is a public label or an entry point name of a procedure to be traced. The "." preceding **name** is required. **Name** must be found in the debug information in the module or **name** must be a public symbol as found in the MAP file. If debug information is used, the address of this tracepoint will be immediately following the prologue of the procedure. If MAP information is used, this address points to the opcode at the given label.

  If the procedure was compiled with debug support, **Name** is case sensitive. If not, C language functions will be case sensitive and begin with an underscore "_" character unless the function is declared with the Pascal calling convention, in which case the underscore is omitted and the name is capitalized.

offs
: (optional) is a decimal (specified as nnnnnnnn) or hex (specified as 0xnnnnnnnn) offset from the entry point address.

RETEP
: (optional) specifies that the tracepoint will be inserted at the **return** address corresponding to this entry point. This is just before the procedure epilogue is executed and at this point the procedure's automatic data is still addressable from register (E)BP and the return code (if any) is set up in (E)AX.

  The module must include information supplied by the debug compile option (see Symbolic Debug Support), meaning that the source language must have been C, otherwise an error message will be generated and this tracepoint discarded.

  When the RETEP is used, the **name** must be a valid entry point to a procedure.

  Note: The RETEP option depends upon the manner in which a C compiler generates its code. Therefore this option may not work will some of the new compilers.

Note: For ASM functions to accomplish tracing, a label must be made public to have a tracepoint applied. Therefore, to accomplish "POST" tracing, a label must be made public at the return statement.

The following are partial examples of Pre/Post tracing of DosOpen:

```
TRACE   MINOR=0x0001,
        TP=.DosOpen,.....            /* Pre-invocation tracing */

TRACE   MINOR=0x8001,
        TP=.DosOpen,RETEP,.....      /* Post-invocation tracing */
```

It is not possible to set dynamic tracepoints on the following machine instructions:

0x9C PUSHF
0xCC INT 3

0xCD	INT n
0xCE	INTO
0x62	BOUND
0x69	IMUL
0x6B	IMUL
0xF6	DIV, IDIV, MUL, IMUL, NEG, NOT, TEST (immediate)
0xF7	DIV, IDIV, MUL, IMUL, NEG, NOT, TEST (immediate)

TRCUST gives an error for these opcodes and the tracepoint is rejected.

In all cases, two tracepoints cannot be applied at the same address.

OPCODE Keyword

The OPCODE parameter is an optional keyword parameter.

```
OPCODE=0xnn,
```

where:

nn is the expected one byte hex opcode to be found at the tracepoint address and TRCUST verifies the value with that in the module. The opcode of the instruction being traced must be the same as this value or an error message is issued and the tracepoint is rejected. This may be used to verify the opcode expected at the address specified by the TP parameter. This may be useful when using `TP = @filename,linenum` to ensure the requested instruction is traced.

TYPE Keyword

The TYPE parameter is an optional keyword parameter that defines the event types of this tracepoint. For more description and examples of event types see the online help for the TRACE command.

```
TYPE=(typename[,typename...]),
```

where:

typename is an ASCII string specifying the type of this tracepoint. The typename symbol must have been previously defined by the TYPELIST statement. If an invalid typename is given, the tracepoint will be discarded and a message issued.

The final type value is obtained by logically combining each type name value using the OR operator. If TYPE is omitted, the trace statement will have a type value of 0.

GROUP Keyword

The GROUP parameter is an optional keyword parameter that defines the group this tracepoint belongs to. For more description and examples of groups see the online help for the **trace** command.

```
GROUP=groupnam,
```

where:

groupnam is an ASCII string specifying which group this tracepoint belongs. The groupname symbol must have been previously defined by the GROUPLIST statement. If an invalid groupname is given, the tracepoint will be

discarded and a message issued.

If GROUP is omitted, the trace statement will have a group value of 0.

DESC Keyword

The DESC parameter is used to produce a description for the tracepoint that is output as the first line of formatted data. It should include the entry point name of the procedure being traced and whether this is an entry or return point. The descriptive string is enclosed in double quotes as for a C language string. The DESC parameter is required if any FMT specifications are present.

The recommended formats for such strings are as follows:

```
DESC="name Pre-Invocation",

DESC="name Post-Invocation",
```

where:

name	is the system component (in parentheses) followed by the entry point name of the procedure.
Pre-Invocation	identifies this tracepoint as an entry point, that is, before the function has been executed.
Post-Invocation	identifies this tracepoint as a return point from the function.

The words **Pre-Invocation** and **Post-Invocation** are not mandatory, merely recommendations to be compatible with the base OS/2 tracepoints, when formatted. If a tracepoint is inserted in the middle of a procedure it will be appropriate to use different wording. The Trace Customizer does not check the wording.

The following is an example of pre-invocation and post-invocation tracepoints:

```
TRACE  MINOR=0x0001,
       TP=.DosOpen,
       DESC="(OS) DosOpen    Pre-Invocation",.....

TRACE  MINOR=0x8001,
       TP=.DosOpen,RETEP,
       DESC="(OS) DosOpen    Post-Invocation",.....
```

FMT Keyword

The optional FMT parameter is used to produce the formatting string for the trace data. The developer should use these to control formatting the output produced by the Trace Formatter. Each **FMT** keyword causes CR/LF to be appended to the format string. The formatting string is similar to a C library printf string specification. It consists of ASCII characters and formatting controls enclosed in double quotes as for a C language string. Each formatting primitive describes the format of the data in the trace buffer at the formatting position and must match the data stored in the trace buffer by the data statements described later. See Formatting Trace Data for a description of how the data is stored in the trace buffer and subsequently formatted.

The formatting controls are as follows:

%Innn — Ignore nnn number of bytes in the trace buffer.

This tells the Trace Formatter to skip over the next nnn bytes in the current trace record. This could be used, for example, to skip over unimportant data, traced as a block, and only output the data of interest.

When using this control, nnn represents an ASCII decimal number and must be followed by a space.

```
statement: FMT = "ignore ten bytes %I10 here",
           FMT = "       and two more %I2 here",
```

```
generates: ignore ten bytes here
                  and two more here
```

%P — Process the data prefix bytes associated with the trace data.

This tells the Trace Formatter that the next bytes in the trace record are the prefix or header bytes for data logged by the dynamic tracing mechanism. This is required to precede any format control describing data logged from memory. Do not use this before data that was logged from a register and never use with static tracepoints. See Formatting Trace Data for a description of how the data is stored in the trace buffer and the use of this control.

```
statements: FMT="memory byte = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69B",

generates:  memory byte = C2
```

%B — Output a byte of data.

```
statement: FMT = "memory byte = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69B"

generates: memory byte = 01
```

%W — Output a word of data.

```
statement: FMT = "register word = %W"

generates: register word = 0001

statement: FMT = "memory word = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W"

generates: memory word = 0001
```

%D — Output a double word of data.

```
statement: FMT = "double word EAX = %D"

generates: double word EAX = 0000 4B2C

statement: FMT = "double memory word = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69D"

generates: double memory word = 0000 4B2C
```

%F — Output a Flat (0:32 bit) address.

```
statement: FMT = "flat address EAX = %F"

generates: flat address EAX = 00004B2C
```

%Q — Output a quad word of data.

```
statement: FMT = "quad word from regs EAX and EBX = %Q"

generates: quad word from regs EAX and EBX = 00004B2C 00000001
```

%A — Output a segmented (16:16 bit) address.

```
statement: FMT = "segmented address in SS:SP = %A"

generates: segmented address in SS:SP = 00B7:0001
```

```
statement: FMT = "segmented address in memory = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.6

generates: segmented address in memory = 00B7:0001
```

%R — Repeat the following format control for the rest of the memory that was logged.

This is used for formatting variable length records. Use this in place of the prefix parameter %P to log the rest of the record in the format specified following the repeat code.

```
statement: FMT = "log a variable number of words from memory = 1W"

generates: log a variable number of words from memory = 0001 0004
```

%S — Output an ASCIIZ string.

The prefix formatting control should always precede this for dynamic tracepoints because the data was logged from memory.

Note: If the tracepoint is static, then **%P** should not be used because the string is terminated with a null byte.

```
statement: FMT = "string = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S"

generates: string = c:\os2\os2.ini
```

%U — Format the remainder of the trace record as a sequence of bytes.

This will output the remaining of the traced data, including any prefix bytes.

```
statement: FMT = "garbage = %U"

generates: garbage = 00 00 00 03 c2 c1 c4 ff 04 00 09 c0 18
```

%X — Output the major event code.

```
statement: FMT = "major code = %X"

generates: major code = 00C2
```

%Y — Output the minor event code.

```
statement: FMT = "minor code = %Y"

generates: minor code = 0081
```

To avoid conflicts with source file control information, all formatting specifications can be in upper or lower case. Also, prefix format specifications may be combined with data format specifications. For example, the following create the same format controls in the TFF:

```
FMT = "/usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W here"
FMT = "%p%w here"
FMT = " %P %W here"
```

LEN Keyword

The LEN parameter is an optional keyword parameter that defines the length of the variable length record that will follow in the next MEM or

MEM32 statement.

```
LEN=(length_spec,flag),
```

where:

length_spec — is an address specification that points to the one word length field of the next memory specification. This format can be `symbolic_name+nnnnnnnn` where `symbolic_name` is a symbolic memory location and `nnnnnnnn` is the offset from that symbolic address. The *length_spec* can also be Flat Register Form or Segment Register Form.

flag — is a mandatory parameter that identifies the level of indirection to be used on the length_spec. The *flag* may be one of the following:

```
D[IRECT]
I[NDIRECT][*[{+|-}iiiiiiii]]...
```

DIRECT implies that the *length_spec* specifies a memory location that contains the length of the variable length record.

INDIRECT means that the *length_spec* contains an address and is dereferenced to obtain the memory location. The optional asterisks denote the level of indirection, one for each level. The indirect offsets `iiiiiiii` are added to or subtracted from the value found at the given level of indirection.

The following are example LEN statements followed by the memory statement whose length they describe.

```
TRACE MINOR=.....,
      /* Symbol vrecord is a record whose first field is a one */
      /* word value that is the total length of the entire     */
      /* variable length record.                               */
      LEN=(vrecord,DIRECT),
      MEM=(.vrecord,DIRECT,LEN),

      /* Symbol vrec_ptr is a pointer to a variable length record */
      /* and vend_ptr is a pointer to the end of the same record. */
      /* The second field (10 bytes from end of record) is  total */
      /* length of the variable length record.                    */
      LEN=(vend_ptr,INDIRECT*-10),
      MEM=(.vrec_ptr,INDIRECT,LEN),

      /* Symbol vrec_ptr is a pointer to a variable length record.*/
      /* The second field (2 bytes from beginning of record) is   */
      /* total length of the variable length record.              */
      LEN=(vrec_ptr,INDIRECT*+2),
      MEM=(.vrec_ptr,INDIRECT,LEN),

      /* Symbol ind_ptr is a pointer to a structure.  The third   */
      /* field in the structure (6 bytes from beginning) is a     */
      /* pointer to a variable record.  The fourth field in the   */
      /* variable length record (8 bytes from beginning) is the   */
      /* total length of this variable length record.             */
      LEN=(ind_ptr,INDIRECT*+6*+8),
      MEM=(.ind_ptr,INDIRECT*+6*,LEN),

      /* If DS:DI contains the address of ind_ptr, to perform */
      /* the above logging, the statements would be:          */
      LEN=(RDS+DI,INDIRECT*+6*+8),
      MEM=(RDS+DI,INDIRECT*+6*,LEN)
```

DATA_STMT

There are three types of data that may be traced as part of the optional DATA_STMT section of the TRACE statement.

Registers

Memory
ASCIIZ strings

More than one keyword is permitted in a tracepoint definition. The order of the statements defines the order in which the data is inserted into the trace buffer.

The combined amount of data to be traced for a single tracepoint cannot exceed MAXDATALENGTH. If TRCUST determines that the maximum data size might be exceeded, a warning message is issued but the tracepoint definition will remain valid.

The keywords for tracing the three types of data are REGS, MEM32, MEM, ASCIIZ32, and ASCIIZ.

The REGS keyword identifies which registers are to be recorded in the trace buffer.

The MEM32 keyword is used to record sections of memory in the trace buffer. Access to this memory location is through 32-bit flat addresses from functions compiled using 32-bit addressing. Several MEM32 parameters may be coded at any one tracepoint if several different memory areas are to be traced.

The MEM keyword is also used to record sections of memory in the trace buffer, but access to this memory is through a segment:offset pair. This is used for functions compiled using 16-bit addressing with segment registers. Several MEM parameters may be coded at any one tracepoint if several different memory areas are to be traced.

The ASCIIZ32 keyword is used to record an ASCIIZ string in the trace buffer. This is a special form of the MEM32 keyword and there may be more than one ASCIIZ32 parameter coded for a single tracepoint.

The ASCIIZ keyword is used to record an ASCIIZ string in the trace buffer. This is a special form of the MEM keyword and there may be more than one ASCIIZ parameter coded for a single tracepoint.

REGS Keyword

This is coded as:

```
REGS=(register[,register]...),
```

where:

register — is one of the following to support OS/2 versions 1.1 and 1.2:

CS, DS, SS, ES, AX, BX, CX, DX, SP, BP, SI, DI, IP, FLAGS

with the addition of the following to support OS/2 version 2.0:

EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI, EFLAGS, EIP, FS, GS

or the symbolic name of a C language variable declared with the register storage-class specifier as `.symbolic_name`.

The same register may appear multiple times in the register list. It will be traced as many times as it appears. Extended registers (E) are 32 bits and logged as two words. All other registers are 16 bits and logged as one word.

Note: To log a C language variable declared with the register storage class, debug information must exist and the variable name is case sensitive. When formatting the data logged from a register variable, remember that there are no memory prefix bytes put into the log buffer.

The following is an example of the REGS statement:

```
/* Given the following declaration in a C language source file: */
register int ret_code;

/* To log registers AX, CX and the register variable ret_code: */
TRACE MINOR=.....
      REGS=(AX,CX,.ret_code),
      FMT="AX=%W  CX=%W  ret_code=%W"
```

MEM32 Keyword

This is used to log memory in a function compiled using 32-bit flat addressing and is coded as:

```
MEM32=(address_spec,flag,{length|LEN}),
```

where:

address_spec	is a flat memory address specification as described in Address Specification.

flag	is a mandatory parameter that identifies the level of indirection to be used on the address. The *flag* may be one of the following:
	
	`D[IRECT]`
	`I[NDIRECT][*[{+|-}iiiiiiii]]...`
	`IS`
	
	DIRECT implies that the address specifies a memory location to be saved in the trace buffer.
	
	INDIRECT means that the address contains a flat address and is dereferenced to obtain the memory location. The optional asterisks denote the level of indirection, one for each level. The indirect offsets **iiiiiiii** are added to or subtracted from the value found at the given level of indirection.
	
	IS (Indirect Segmented) means that the address contains a segmented address that is dereferenced to obtain the memory location.

length	is the number of bytes at the memory location to be saved in the trace buffer. If *length* is too big, a warning message will be given, and *length* will be set to MAXDATALENGTH. If *length* is 0 an error message will be given, and this tracepoint will be ignored.
LEN	specifies that this is a variable length record to log and the length was specified by the preceding LEN statement. If there was no preceding LEN statement, this tracepoint is rejected. Either length or LEN must be specified, but not both.

The following is an example of the MEM32 statement:

```
TRACE MINOR=.....
      /* To log retcode enter the following: */
      MEM32=(.retcode,DIRECT,2),

      /* s_ptr is a pointer to a structure, log it for 4 bytes. */
      MEM32=(.s_ptr,INDIRECT,4),

      /* Field 6 bytes into structure pointed at by s_ptr is a     */
      /*  pointer to a structure, log 8 bytes past begin of struct.*/
      MEM32=(.s_ptr,INDIRECT*+6*+8,10),  /* logs ten bytes */

      /* s_ptr points to a variable length record, second field */
      /*  is the record length (offset 4 from record beginning).*/
      LEN=(s_ptr,INDIRECT*+4),
      MEM32=(.s_ptr,INDIRECT,LEN)

      /* s_end points to the end of same variable length record,*/
      /*  second field is the record length (offset -6 from     */
      /*  record beginning).                                    */
      LEN=(s_end,INDIRECT*-6),
      MEM32=(.s_ptr,INDIRECT,LEN)
```

MEM Keyword

This is used to log memory in a function compiled using 16-bit segment:offset addressing and is coded as:

```
MEM=(address_spec,flag,{length|LEN}),
```

where:

address_spec — is a segmented memory address specification as described in Address Specification.

flag — is a mandatory parameter that identifies the level of indirection to be used on the address. The *flag* may be one of the following:

```
D[IRECT]
I[NDIRECT][*[{+|-}iiiiiiii]]...
IF
```

DIRECT implies that the address specifies a memory location to be saved in the trace buffer.

INDIRECT means that the address contains a segmented address and is dereferenced to obtain the memory location. The optional asterisks denote the level of indirection, one for each level. The indirect offsets `iiiiiiii` are added to or subtracted from the value found at the given level of indirection.

IF (Indirect Flat) means that the address contains a flat address that is dereferenced to obtain the memory location.

Only far pointers may be dereferenced when using segmented addressing.

length — is the number of bytes at the memory location to be saved in the trace buffer. If *length* is too big, a warning message will be given, and *length* will be set to MAXDATALENGTH. If *length* is 0 an error message will be given, and this tracepoint will be ignored.

LEN — specifies that this is a variable length record to log and the length was specified by the preceding LEN statement. If there was no preceding LEN statement, this tracepoint is rejected. Either length or LEN must be specified, but not both.

ASCIIZ32 Keyword

This is used to log a string in a function compiled using 32-bit flat addressing and is coded as:

```
ASCIIZ32=(address_spec,flag,maxlength),
```

where:

address_spec — is a 0:32 bit flat memory address specification as described in Address Specification.

flag — is a mandatory parameter that identifies the level of indirection to be used on the address. The *flag* may be one of the following:

```
D[IRECT]
I[NDIRECT][*[{+|-}iiiiiiii]]...
IS
```

DIRECT implies that the address points to a memory location, the contents of which are to be saved in the trace buffer.

INDIRECT means that the address points to a flat address pointer which is dereferenced to obtain the target location to save in the trace buffer. The optional asterisks denote the level of indirection, one for each level. The indirect offsets `iiiiiiii` are added to or subtracted from the value found at the given level of indirection.

IS (Indirect Segmented) means that the address points to a segmented address pointer which is dereferenced to obtain the target location to save in the trace buffer.

maxlength — is the maximum length of the string that will be saved in the trace buffer. It should be no greater than MAXDATALENGTH. The actual length to be traced will depend on where the zero terminating byte is found.

If *maxlength* is 0 an error message will be given, and this tracepoint will be ignored.

Note: When using dynamic tracing, the OS/2 kernel does not place the terminating null byte into the trace buffer; therefore the prefix byte must be used by the Trace Formatter to obtain the length of the string.

ASCIIZ Keyword

This is used to log a string in a function compiled using 16-bit segment:offset addressing and is coded as:

```
ASCIIZ=(address_spec,flag,maxlength),
```

where:

address_spec — is a segmented memory address specification as described in Address Specification.

flag — is a mandatory parameter that identifies the level of indirection to be used on the address. The *flag* may be one of the following:

```
D[IRECT]
I[NDIRECT][*[{+|-}iiiiiiii]]...
IF
```

DIRECT implies that the address points to a memory location, the contents of which are to be saved in the trace buffer.

INDIRECT means that the address points to a far pointer which is a segmented address that is dereferenced to obtain the target location to save in the trace buffer. The optional asterisks denote the level of indirection, one for each level. The indirect offsets `iiiiiiii` are added to or subtracted from the value found at the given level of indirection.

IF (Indirect Flat) means that the address points to a far pointer, which is a flat address that is dereferenced to obtain the target location to save in the trace buffer. Only far pointers may be dereferenced using segmented addresses.

maxlength — is the maximum length of the string that will be saved in the trace buffer. It should be no greater than MAXDATALENGTH. The actual length to be traced will depend on where the zero terminating byte is found.

If *maxlength* is 0 an error message will be given, and this tracepoint will be ignored.

Note: When using dynamic tracing, the OS/2 kernel does not place the terminating null byte into the trace buffer; therefore the prefix byte must be used by the Trace Formatter to obtain the length of the string.

Address Specification

The syntax for specifying a memory address given here applies to the MEM32, MEM, ASCIIZ32 and ASCIIZ keywords above.

An address is specified in one of the following forms:

- Symbolic name form (can be used for MEM32, MEM, ASCIIZ32, and ASCIIZ).
- Flat register form (can be used only for MEM32 and ASCIIZ32).
- Segment register form (can be used only for MEM and ASCIIZ).

Each of these forms is described below.

Symbolic Name Form

This is coded as:

```
.name[{+|-}nnnnnnnn]...[{+|-}(iiiiiiii)],
```

where:

name	is a symbolic name of a memory location. The "." is required before the **name** The debug information in the module is checked for the **name**. If the debug information for the name is not found and a MAP was given, the MAP is checked. An error message is output by the Trace Customizer if the symbol is not found and the trace definition is ignored. The **name** is case sensitive except under the conditions that follow: If the procedure containing **name** was not compiled with debug option then if **name** is a C language symbolic name it will be case sensitive and begin with an underscore (_) character unless it was declared with the Pascal naming convention, in which case the underscore is omitted and **name** is capitalized.
nnnnnnnn	(optional) is a displacement from the symbolic address. If hex the syntax is 0xnnnnnnnn.
iiiiiiii	(optional) is a displacement from the indirect address. If hex the syntax is 0xiiiiiiii. This specifies a displacement from the final address when using INDIRECT, IF (Indirect Flat) or IS (Indirect Segmented) addressing.

Flat Register Form

This is coded as:

```
Fbreg[{+|-}ireg]...[{+|-}nnnnnnnn]...[{+|-}(iiiiiiii)],
```

where:

breg	is a flat model (0:32 bit) base register and is one of: EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI
ireg	(optional) is an extended data, base or index register. More than one ireg may be used to define a displacement from the flat register value to the memory location. It may be one of: EAX, EBX, ECX, EDX, EBP, ESI, EDI
nnnnnnnn	(optional) is an optional fixed displacement to be added to the address calculated in the registers. If hex the format is 0xnnnnnnnn.
iiiiiiii	(optional) is a displacement from the indirect address. If hex the syntax is 0xiiiiiiii. This specifies a displacement from the final address when using INDIRECT or IS (Indirect Segmented) addressing.

This form of address is calculated at run time.

Segment Register Form

This is coded as:

```
R[sreg][{+|-}dreg]...[{+|-}nnnnn]...[{+|-}(iiiii)],
```

where:

sreg	is a segment register and is one of: CS, DS, SS, ES, FS, GS
dreg	(optional) is a data, base or index register. More than one dreg may be used to define a displacement from the segment register value to the memory location. It is one of: BP, SP, SI, DI, AX, BX, CX, DX
nnnnn	(optional) is an optional fixed displacement to be added to the address calculated in the registers. If hex the syntax is 0xnnnn.
iiiii	(optional) is a displacement from the indirect address. If hex the syntax is 0xiiii. This specifies a displacement from the final address when using INDIRECT or IF (Indirect Flat) addressing.

This form of address is calculated at run time.

Formatting Trace Data

This section gives a brief description of the formatting process as an aid to generating correct formatting strings.

Each trace record stored in the RAS buffer consists of a header followed by a number of variable length trace data records. The header identifies the major and minor code, time stamp, process ID, etc., and the total length of the trace data for that trace record.

Each MEM32, MEM, ASCIIZ32, or ASCIIZ data statement, coded in the trace source file for a tracepoint, produces an associated data record to be stored in the trace buffer. The data records consist of a 3-byte prefix followed by the trace data. This prefix consists of a status byte followed by the length of the data for that statement. The status byte indicates whether valid data has been traced.

Dynamic trace can only trace data that is resident in memory at the time that the tracepoint is executed. Data may not be able to be traced for two reasons: it resides in a page that is currently paged out or the address specified is invalid. This latter case usually occurs due to tracing indirectly via invalid pointer variables. In either of these two cases dynamic trace sets the status byte accordingly and stores the pointer in the place of the wanted data. No more data is attempted to be traced for this invocation of the tracepoint, but tracing will resume the next time this tracepoint is encountered.

Since the position of these prefix bytes, within a trace record, is dependent on the data being traced and the number of MEM32, MEM, ASCIIZ32, or ASCIIZ statements, the Trace Formatter must be told when to expect the prefix in the trace record. This is the purpose of the **%P** formatting control. It must be coded in the formatting string at every place a data record is expected.

Note: With ASCIIZ and ASCIIZ32 commands, the prefix must be used to obtain the length of the string since the string is not null terminated.

Sample Trace Source Files

This section gives four sample TSF files. The first is for a module written in a mix of C and MASM and compiled with 16:16 segmented addressing. The second was compiled with 0:32 flat addressing. The third module consists of routines, some which were compiled using 16-bit segmented addressing and some that were compiled using 32-bit flat addressing. The fourth is for monitoring function references in a module.

TSF Using 16-bit Segmented Addressing

```
;  Trace source file for the xxx dynalink.  Compiled with 16-bit offsets.

     MODNAME=\c\src\xxx.dll
     MAJOR=0xC5
     MAXDATALEN=200
;  We will want to trace up to 200 bytes in any one trace call.

     TYPELIST NAME=API,ID=08,
              NAME=SYS,ID=04,
              NAME=PRE,ID=02,
              NAME=POST,ID=64

     GROUPLIST NAME=MEM,ID=1,
               NAME=FS,ID=3

              /* The following tracepoint does not need debug info,
                 only a MAP file is necessary with label xxalloc
                 public in it.  The program must be compiled in 16-bit
                 mode because segmented addressing is used (ASCIIZ
                 instead of ASCIIZ32).
                 This logs the word registers AX and BX and the string
                 pointed at by DS:DI for a max of 20 bytes. */

     TRACE    MINOR=25, TP=.xxalloc,
              OPCODE=0x8B,    /* the opcode is optional */
              TYPE=(API,PRE),
              GROUP=MEM,
              DESC="(OS) xxalloc Pre-Invocation",
              FMT ="                              AX = %W ",
              FMT ="                              upper BX = %B",
              FMT ="                              lower BX = %B",
              FMT ="                              param = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S",
              REGS=(AX,BX),
              ASCIIZ=(RDS+DI,DIRECT,20)


              /* This defines a tracepoint at Foo label.  The ten words
                 to log are found indirectly through SS:SP.  Note that
                 each word needs a format control but since only one
                 memory access was done,  one prefix control is needed. */

      TRACE   MINOR=0xB0, TP=.Foo,
              TYPE=(SYS),
              GROUP=FS,
              DESC="(OS) Foo Pre-Invocation",
              FMT="                     First Five words = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69Wtrtrcust.scr
              FMT="                     Three words ignored %I6",
              FMT="                     Last Two Words = trtrcust.scr, tk.sdb.ToolsReference, id207sW",
              MEM=(RSS+SP,INDIRECT,20)


              /* This defines a tracepoint at Goo label.  DS:DI points
                 to a structure whose second field is a pointer to an
                 ASCIIZ string.  The offset from the first field in the
                 structure is 4 bytes. Max string size to log is 40 bytes. */

      TRACE   MINOR=0xB1, TP=.Goo,
              TYPE=(SYS),
              GROUP=FS,
              DESC="(OS) Goo Pre-Invocation",
              FMT="          Second field in struct points to /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S",
              ASCIIZ=(RDS+DI+4,INDIRECT,40)


              /* This defines a tracepoint at Hoo label.  DS:DI points to
                 memory that contains a pointer to a structure. We want to
                 log the third field in the structure (offset 6 from begin
                 of structure). */

      TRACE   MINOR=0xB2, TP=.Hoo,
              TYPE=(SYS),
              GROUP=FS,
              DESC="(OS) Hoo Pre-Invocation",
              FMT="          Third field in struct is doubleword = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69D",
              MEM=(RDS+DI,INDIRECT*+6,4)
```

```
        /* This defines a tracepoint at Zoo label.  DS:DI points to
           memory that contains a pointer to end of a structure. We
           want to log the last field in the structure(offset -2 from
           end of structure). */

TRACE   MINOR=0xB3, TP=.Zoo,
        TYPE=(SYS),
        GROUP=FS,
        DESC="(OS) Zoo Pre-Invocation",
        FMT="          Last field in struct is word = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W",
        MEM=(RDS+DI,INDIRECT*-2,2)


        /* This defines a tracepoint at procedure CheckIT.  This
           is a C routine compiled with debug information.  The
           data to log is an ASCIIZ string called NameIt.  */

TRACE   MINOR=0xB3, TP=.CheckIt,
        TYPE=(PRE),
        GROUP=FS,
        DESC="(OS) CheckIt Pre-Invocation",
        FMT="              NameIt = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S",
        ASCIIZ=(.NameIt,DIRECT,64)


        /* This defines a tracepoint at the return point of the
           procedure CheckIt, a C routine compiled with debug.
           Status_Rec is a record variable. We want to log the
           age field (four bytes from the begin of Status_Rec),
           the name (six bytes from Status_Rec that points to
           an ASCIIZ string), the age of the next Status_Rec
           (a pointer to the next Status_Rec is ten bytes from
           the begin of Status_Rec, the age is four bytes from
           the begin of the next Status_Rec). */

TRACE   MINOR=0x80B3, TP=.CheckIt,RETEP,
        TYPE=(POST),
        GROUP=FS,
        DESC="(OS) CheckIt Post-Invocation",
        FMT="              Status_Rec.age = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W",
        FMT="              Status_Rec.name = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S",
        FMT="              Status_Rec.next->age = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W",
        MEM=(.Status_Rec+4,DIRECT,2),
        ASCIIZ=(.Status_Rec+6,INDIRECT,64),
        MEM=(.Status_Rec+10,INDIRECT*+4,2)


        /* This defines a tracepoint at line 58 in the source
           file check.c   Debug info is needed to use this
           type of tracepoint. v_ptr is a pointer to a variable
           sized record.  The length is 4 bytes past the
           beginning of the record.  Log that record. */

TRACE   MINOR=0x71B4, TP=@check.c,58,
        TYPE=(SYS),
        GROUP=FS,
        DESC="(OS) CheckIt   before allocation",
        FMT="              Variant Record = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W96/09/03U",
        LEN=(v_ptr,INDIRECT*+4),
        MEM=(.v_ptr,INDIRECT,LEN)


        /* This does not define a tracepoint, it only defines a
           trace formatting string for minor code 181 (B5 hex). */

TRACE   MINOR=0xB5, TP=@STATIC,
        DESC="(OS) StaticProcedure Pre-Invocation",
        FMT="                 DI = %W FLAGS = %W"


        /* This defines a tracepoint at routine LookUp, but no
           data is to be logged, only the DESC will show up
           in the Trace log when the tracepoint is formatted. */

TRACE   MINOR=0xB6, TP=.LookUp,
        TYPE=(SYS),
        GROUP=FS,
```

```
            DESC="(APP) LookUp  Pre-Invocation",
```

TSF Using 32-bit Addressing

```
;  Trace source file for the NEW dynalink.  Compiled with 32-bit offsets.

     MODNAME=NEWCALLS.DLL
     MAJOR=241
     MAXDATALEN=200
;  We will want to trace up to 200 bytes in any one trace call.

     TYPELIST NAME=API,ID=08,
              NAME=SYS,ID=04,
              NAME=PRE,ID=02,
              NAME=POST,ID=64

     GROUPLIST NAME=MEM,ID=1,
               NAME=FS,ID=3

              /* The following tracepoint does not need debug info,
                 only a MAP file is necessary with label NewAllocSeg
                 public in it.  The program must be compiled in 32-bit
                 mode because flat addressing is used (ASCIIZ32 instead
                 of ASCIIZ).
                 This logs lower word of EAX, the double word of EBX
                 and the string at the address specified by ESP with
                 offset ESI. */

     TRACE    MINOR=45, TP=.NewAllocSeg,
              TYPE=(API,PRE),
              GROUP=MEM,
              DESC="(NEW) NewAllocSeg Pre-Invocation",
              FMT ="                              AX = %W ",
              FMT ="                              EBX = %F",
              FMT ="                              param = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S",
              REGS=(AX,EBX),
              ASCIIZ32=(FESP+ESI,DIRECT,20)

              /* This defines a tracepoint at Foo label.  The ten words
                 to log are found indirectly by using EBP with offset
                 EDI. Note that each value logged needs a format control. */

      TRACE   MINOR=0xD0, TP=.Foo,
              TYPE=(SYS),
              GROUP=FS,
              DESC="(NEW) Foo Pre-Invocation",
              FMT="                     First Five words = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69Wtrtrcust.scr
              FMT="                     Three words ignored %I6",
              FMT="                     Last Two Words = trtrcust.scr, tk.sdb.ToolsReference, id207sW",
              MEM32=(FEBP+EDI,INDIRECT,20)


              /* This defines a tracepoint at Goo label. EAX + EDI points
                 to a structure whose second field is a pointer to an
                 ASCIIZ string.  The offset from the first field in the
                 structure is 4 bytes. Max string size to log is 40 bytes.*/

      TRACE   MINOR=0xD1, TP=.Goo,
              TYPE=(SYS),
              GROUP=FS,
              DESC="(NEW) Goo Pre-Invocation",
              FMT="           Second field in struct points to /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S",
              ASCIIZ32=(FEAX+EDI+4,INDIRECT,40)


              /* This defines a tracepoint at Hoo label.  EBP + EDI points
```

```
           to memory that contains a pointer to a structure. We want
           to log the third field in the structure (offset 6 from
           begin of structure). */

TRACE   MINOR=0xD2, TP=.Hoo,
        TYPE=(SYS),
        GROUP=FS,
        DESC="(NEW) Hoo Pre-Invocation",
        FMT="           Third field in struct is doubleword = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69D",
        MEM32=(FEBP+EDI,INDIRECT*+6,4)


        /* This defines a tracepoint at Zoo label.  EAX + EDI points
           to memory that contains a pointer to end of a structure. We
           want to log the last field in the structure (offset -2 from
           end of structure). */

TRACE   MINOR=0xD3, TP=.Zoo,
        TYPE=(SYS),
        GROUP=FS,
        DESC="(OS) Zoo Pre-Invocation",
        FMT="          Last field in struct is word = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W",
        MEM=(FEAX+EDI,INDIRECT*-2,2)


        /* This defines a tracepoint at procedure CheckIT.  This is
           a C routine compiled with debug information.  The
           data to log is an ASCIIZ string called NameIt. */

TRACE   MINOR=0xD3, TP=.CheckIt,
        TYPE=(PRE),
        GROUP=FS,
        DESC="(NEW) CheckIt Pre-Invocation",
        FMT="              NameIt = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S",
        ASCIIZ32=(.NameIt,DIRECT,64)


        /* This defines a tracepoint at the return point of the
           procedure CheckIt, a C routine compiled with debug.
           Status_Rec is a record variable. We want to log the
           age field (four bytes from the begin of Status_Rec)
           the name (six bytes from Status_Rec that points to
           an ASCIIZ string) and the age of the next Status_Rec
           (a pointer to the next Status_Rec is ten bytes from
           the begin of Status_Rec, the age is four bytes from
           the begin of the next Status_Rec). */

TRACE   MINOR=0x80D3, TP=.CheckIt,RETEP,
        TYPE=(POST),
        GROUP=FS,
        DESC="(NEW) CheckIt Post-Invocation",
        FMT="              Status_Rec.age = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W",
        FMT="              Status_Rec.name = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69S",
        FMT="              Status_Rec.next->age = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W",
        MEM32=(.Status_Rec+4,DIRECT,2),
        ASCIIZ32=(.Status_Rec+6,INDIRECT,64),
        MEM32=(.Status_Rec+10,INDIRECT*+4,2)


        /* This does not define a tracepoint, it only defines a
           trace formatting string for minor code 223 (DF hex). */

TRACE   MINOR=0xDF, TP=@STATIC,
        DESC="(NEW) StaticProcedure Pre-Invocation",
        FMT="                DI = %W FLAGS = %W"


        /* This defines a tracepoint at routine LookUp, but no
           data is to be logged, only the DESC will show up
           in the Trace log when the tracepoint is formatted.
           LookUp is a C language routine not compiled with
           debug and not declared with Pascal
           calling conventions; the underscore is needed for
           this label. */

TRACE   MINOR=0xE0, TP=._LookUp,
        TYPE=(SYS),
        GROUP=FS,
        DESC="(NEW) LookUp  Pre-Invocation"
```

TSF Using Mix of 16-bit and 32-bit Addressing

```
;  Trace source file for the MIXED dynalink.
;    Parts were compiled with 16-bit compiler, some with 32-bit compiler.
;    The developer must know how the parameters being sent in are
;    to be addressed, whether they are segmented or flat addresses.

     MODNAME=MIXCALLS.DLL
     MAJOR=250
     MAXDATALEN=200
;  We will want to trace up to 200 bytes in any one trace call.

     TYPELIST NAME=API,ID=08,
              NAME=SYS,ID=04,
              NAME=PRE,ID=02,
              NAME=POST,ID=64

     GROUPLIST NAME=MEM,ID=1,
               NAME=FS,ID=3

              /* The following tracepoint is for the routine MixStub.
                 This was compiled using segmented addressing and
                 one of the parameters to it is a pointer to a control
                 block called mix_ctrl.  This pointer, found at SS:SP,
                 is a flat address because the routine that sent it was
                 compiled with the flat addressing specification.
                 This logs the mix_ctrl block for 6 bytes. */

     TRACE    MINOR=95, TP=.MixStub,
              TYPE=(API,PRE),
              GROUP=MEM,
              DESC="(OS) MixStub     Pre-Invocation",
              FMT ="                              mix_ctrl = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W %W %W",
              MEM=(RSS+SP,IF,6)    /* is an indirect flat address */


              /* The following is for the routine FlatStub.  This was
                 compiled using 32-bit flat addresses.  A parameter to
                 flatstub is a pointer called p_seg_info.  This
                 pointer is a segmented address because the routine
                 calling flatstub was compiled using 16-bit segmented
                 addressing.  Log where p_seg_info points for 2 bytes. */

      TRACE   MINOR=0xf0, TP=.FlatStub,
              TYPE=(SYS),
              GROUP=FS,
              DESC="(OS) FlatStub Pre-Invocation",
              FMT="                       seg_info = /usr/cmvc/family/pubdoc/vc/0/1/2/2/s.69W",
              MEM32=(.p_seg_info,IS,2)   /* value p_seg_info is a 16-bit */
                                         /*  segmented address           */
```

Trace Customizer Messages

The messages generated by the Trace Customizer are given below. In addition to the message itself, the source line in error is displayed.

Errors in the FATAL and SEVERE category will cause the compiler to abort immediately.

ERROR messages will normally cause a tracepoint definition to be discarded and processing continues with the next definition.

WARNING messages will allow a valid tracepoint definition to be produced, and the results will normally be as expected.

External Messages

Msg No.	Message Text
TCU0001I:	TRCUST Version %1
TCU0002I:	created TDF file %1
TCU0004I:	error detected, TRCUST aborted
TCU0005E:	unrecognized executable header in %1
TCU0006E:	invalid command line parameter %1
TCU0007E:	unable to access %1
TCU0008E:	unable to open %1
TCU0009E:	unable to read from %1
TCU0010E:	unable to write to %1
TCU0011E:	unable to find %1 or access denied
TCU0012E:	out of memory
TCU0013E:	symbolic information not given for %1
TCU0014E:	invalid quoted string encountered in %1
TCU0015E:	too many tracepoints in %1
TCU0016E:	unexpected end-of-file encountered in %1
TCU0017I:	error detected, tracepoint ignored
TCU0018W:	%1 (%2): invalid object number %3 in executable file
TCU0019W:	%1 (%2): invalid offset for object number %3 in executable file
TCU0020W:	%1 (%2): invalid page for tracepoint application
TCU0021W:	%1 (%2): unable to find %3 in CodeView information
TCU0022W:	%1 (%2): invalid line number in %3
TCU0023W:	%1 (%2): invalid opcode for tracing
TCU0024W:	%1 (%2): unable to find %3 symbol
TCU0025W:	%1: duplicate %3 in tracepoint starting at line %2
TCU0026W:	%1 (%2): index too large, high word ignored
TCU0027W:	%1 (%2): invalid %3 syntax
TCU0028W:	%1 (%2): %3 not specified
TCU0029W:	%1 (%2): %3 redefinition

TCU0030W:	%1 (%2): unable to find line number, %3 used
TCU0031W:	%1 (%2): CodeView information does not exist for %3
TCU0032W:	%1 (%2): tracing opcode defined after %3
TCU0033W:	%1 (%2): %3 out of range, default assumed
TCU0034W:	%1 (%2): syntax error, %3 unrecognized
TCU0035W:	%1 (%2): RPN command record exceeds maximum limit

Internal Messages

Internal format messages produced by TRCUST are of the form:

```
TRCUST(n) severity: message_text
```

where:

n	is the line number of the tsf in error or (1) if the command syntax is in error.
severity	is the message severity and may take one of the following values: • Fatal Messages • Severe Messages • Error Messages • Warning Messages
message_text	is the text of the message.

Fatal Messages

Msg No.	Message Text
1	TSF file not specified
2	file not found or access denied : %s
3	cannot open file : %s
4	error accessing file : %s
5	error writing to file : %s
6	unable to allocate more memory
7	too many tracepoints in file
8	error reading file: %s, Rc = %s
9	changing file pointer for: %s, Rc = %s
10	unknown EXE header type for: %s

11	invalid path specified in combine file
12	max TFF files to combine is 50
13	all TFFs not have same major code, file: %s
14	invalid MAP file extension given in: %s
15	TCF file not specified
16	filename to long: %s
17	token in TSF file exceeds %s bytes

Severe Messages

Msg No.	Message Text
33	module name not specified
34	premature end of file encountered
35	syntax error : missing '%s' before '%s'
36	new line in literal
37	NULL in literal
38	keyword '%s' expected, '%s' found
39	symbolic info not given for %s
40	MAJOR redefinition
41	TDFID redefinition
42	MAXDATALENGTH redefinition
43	line too long in input file: %s

Error Messages

Msg No.	Message Text
65	number expected, '%s' found
66	unexpected: %s, ignored
67	minor code not specified
68	minor code out of range
69	TYPELIST redefinition, ignored
70	GROUPLIST redefinition, ignored
71	TP redefinition, tracepoint ignored

72	MINOR redefinition, tracepoint ignored
73	OPCODE redefinition, tracepoint ignored
74	syntax error: missing '%s' before '%s'
75	opcode: %s out of range
76	opcode at TP address cannot be traced
77	opcode mismatch at address to apply TP
78	register expected, '%s' found
79	symbol not found: %s
80	address not found
81	segment register expected, '%s' found
82	trace record incomplete, '%s' required
83	RPN command record exceeds 255 bytes
84	invalid parameter: '%s', ignored
85	invalid ID: %s, ignored
86	group/type redefinition: %s, ignored
87	typeid redefinition: %s, ignored
88	groupid redefinition: %s, ignored
89	invalid address specified: %s
90	line number past end of code for file %s
91	Debug info does not exist for: %s
92	line number missing or invalid: %s
93	filename %s not found in Debug info
94	duplicate minor code = %s, ignored
95	duplicate minor code = %s in file %s, ignored
96	variable LEN parameter not preceding
97	RPN stack limit of 16 exceeded
98	invalid flat register specified: %s
99	total FMT format specs above 4096 bytes
100	zero length specified, tracepoint ignored
101	ORBIT redefinition, tracepoint ignored
102	invalid ORBIT value, tracepoint ignored
103	opcode defined after ABORT
104	opcode defined after REMOVE
105	duplicate TP address, ignored
106	/D not allowed with /C, ignored

Warning Messages

Msg No.	Message Text
129	MAXDATALENGTH out of range, 512 used
130	typename unknown: %s, ignored
131	groupname unknown: %s, ignored
132	file: %s, extension invalid, using: %s
133	'%s' expected before '%s', one assumed
134	too many %s, first 16 types, 48 groups used
135	name too long: %s, first 8 characters used
136	linenum in file: %s not found, using #%s
137	bad object number: %s used for file %s
138	offset %s is invalid for object number %s
139	page tracepoint to be applied at not valid
140	MAXDATALENGTH to log could be exceeded
141	MAJOR out of range, 1 used
142	TDFID out of range, 0 used
143	index too large, high word ignored

Workplace Class List

The Workplace Class List is a tool that creates a workplace object class and an instance of a workplace object class. Workplace objects are constructed using the System Object Model (SOM) protocol and are instances of one of the following workplace object classes:

Predefined	These classes are defined by the system. Examples of predefined workplace object classes are WPObject, WPFileSys, and WPAbstract.
Subclass	These classes are derived from existing predefined workplace object classes. They add, remove, or change functions, but they retain the basic behaviors and capabilities of the parent class.
Replaced	These classes replace the class being *subclassed*. They modify the behavior of an instance of a predefined workplace object class without the instance being aware of the new class.

Starting Workplace Class List

To start Workplace Class List, select the **PM Development Tools** folder, and then select **Workplace Class List**. A window appears. The window contains a list of the workplace object classes currently registered in the OS/2 Workplace Shell. Using the window, you can:

- Create an instance of a workplace object class
- Replace a workplace object class

- Restore a replaced workplace object class
- Add a workplace object class
- Delete a workplace object class

Creating an Object-Class Instance

To create an instance of a workplace object class:

1. Select the class from the list in the Workplace Object Class window.
2. Display the pop-up menu by clicking mouse button 2.
3. Select the **Create Instance** choice.

   Note: Only an instance of a physical workplace object class can be created. In other words, you cannot create instances of WPObject or WPClass because they are not physical classes.
4. Complete the following input fields:

Object Title	The text string you assign to the object. The text string becomes the object title and appears under the object when the object is displayed on the Workplace Shell. When the object is in an opened state, the text string appears in the title bar of the window.
Class of new object	The name of the class of which the object you selected is a member.
Parameters	A series of **keyname=value** pairs (separated by semicolons) that change the behavior of the object. Each object class defines the key names and parameters it expects to see. All parameters have safe defaults, so it is never required that parameters be passed to an object.
Location	A real name specified by a fully qualified file specification, such as C:\OS2\DLL\MINXOBJ.DLL, or a logical name specified by a predefined symbol.

Examples of logical names include the following:

LOCATION_NOWHERE	Hidden folder
LOCATION_DESKTOP	OS/2 Desktop (Workplace)
LOCATION_SYSTEM	OS/2 System folder
LOCATION_TEMPLATES	Template folder
LOCATION_SYSTEMSETUP	System setup folder
LOCATION_STARTUP	Startup folder
LOCATION_INFORMATION	Information folder
LOCATION_DRIVES	Drives folder

Replacing a Workplace Object Class

To replace an existing registered class:

1. Select the class from the list in the Workplace Object Class window.
2. Display the pop-up menu by clicking mouse button 2.
3. Select the **Replace** choice. Note that only classes that have already been registered are valid.
4. Complete the following input fields: **Original class** and **Replacement class**.

Note: The replacement class must be a descendant of the original class. Replacing an object class is useful for modifying the behavior of objects that are instances of the original class but are not aware of the replacement class.

Original class	The name of the object class being replaced in the Workplace.
Replacement class	The name of the object class replacing the original class.

Restoring a Replaced Workplace Object Class

To return a replaced class to its original class:

1. Select the replaced class from the list in the Workplace Object Class window.
2. Display the pop-up menu by clicking mouse button 2.
3. Select the **Unreplace** choice. Note that only classes that have already been replaced are valid.
4. Complete the following input fields:

Original class	The name of the replaced object class being returned to its original object class in the Workplace.
Replacement class	The name of the replaced object class being returned to its original object class.

Adding a Workplace Object Class

To add a class to the Workplace Shell:

1. Display the pop-up menu by clicking mouse button 2.
2. Select the **Add Class** choice.
3. Complete the following input fields:

New class name	The name of object class you want to add to the Workplace. Type the class name exactly as it is built; it is case-sensitive.
Library module	The name of the dynamic link library (DLL) that holds the object definition. Type the library name with complete file specification information. **Note:** The DLL must be created by the IBM System Object Model.

Deleting an Object Class

To delete a class from the Workplace Shell:

1. Select the class you want to delete from the list in the Workplace Object Class window.
2. Display the pop-up menu by clicking mouse button 2.
3. Select the **Delete a Class** choice.
4. Complete the name of the class you want to delete from the Workplace.

Note: You cannot delete classes that are predefined by the operating system, such as WPObject or WPClass.

Error Messages

LINK386 Error Messages

Format of Error Messages

There are three types of LINK386 error messages:

- *Fatal errors* cause LINK386 to stop running. They have the following format: *location* : **fatal error L1***xxx* : *message text*
- *Nonfatal errors* indicate problems in the executable file. LINK386 produces the executable file and sets the error bit in the header for the OS/2 environment. This means that the executable file cannot be run from OS/2. Nonfatal error messages have the following format:

  location : **error L2***xxx* : *message text*
- *Warnings* indicate possible problems in the executable file. LINK386 produces the executable file, but does not set the error bit in the header for the OS/2 environment. Warnings have the following format:

  location : **warning L4***xxx* : *message text*

In all these messages, *location* is the input file associated with the error, or it is LINK386 itself if there is no input file. The *message text* is the actual text message that LINK386 generates. When the input file is a module definition file, the line number is included, as in this example:

```
myfile.def(3): fatal error L1030: missing internal name
```

When the input file is an object file or library file and has a module name, the module name is enclosed in parentheses, as in the following examples:

```
SLIBCE.LIB(_file)
MAIN.OBJ(main.c)
TEXT.OBJ
```

Error Message Descriptions

Fatal Error Messages (Part 1) 1001 - 1049

L1001 *option* : option name ambiguous
Explanation: A unique option name does not appear after the option indicator (/).
Example: The command

```
LINK386 /N main;
```

produces this error because LINK386 cannot tell which of the seven options beginning with the letter N is intended.
Action: Retry using the correct minimum option abbreviation.

L1004 *option* : invalid numeric value
Explanation: An incorrect value appeared for one of the LINK386 options. This might be because a character string has been entered for an option that requires a numeric value or because the proper numeric prefix (for example, 0x for hexadecimal) was not used for a numeric value.
Action: Retry with a numeric value.

L1006 *option* : stack size exceeds 65,535 bytes
Explanation: The size you specified for the stack in the /STACK option of the link command is more than 65,535 bytes.
Action: Retry with a stack size of less than or equal to 65,535 bytes.

L1008 *option* : segment limit set too high
Explanation: The specified limit on the /SEGMENTS option is greater than 16 375.
Action: Retry with a limit in the range 1 to 16 375.

L1020 no object modules specified
Explanation: You did not specify any object file names to the linker.
Action: Restart LINK386, including at least one object file name.

L1021 cannot nest response files
Explanation: A response file has been named within another response file. You have used @*filename* within the response file. The @ symbol is reserved by LINK386 to signify a response file name.
Action: Edit the response file to remove the nested response file.

L1022 response line too long
Explanation: A line in an automatic response file is longer than 256 characters.
Action: Edit the line to make it shorter than 256 characters. Response files can contain more than one line.

L1023 terminated by user
Explanation: You pressed Ctrl+C or Ctrl+Break.
Action: Your action has terminated LINK386. Restart if necessary.

L1030 missing internal name
Explanation: You have not specified an internal name for an import in the module definition file.
Action: Edit the module definition file, giving an internal name so that LINK386 can identify references to the import.

L1031 module description redefined
Explanation: You have used the DESCRIPTION keyword for a module in the module definition file more than once.
Action: Edit the module definition file, deleting the extra descriptions.

L1032 module name redefined
Explanation: You have defined a module name more than once with the NAME or LIBRARY keyword in the module definition file.
Action: Edit the module definition file, checking the module name definitions.

L1033 input line too long; *number* characters allowed
Explanation: The input line contains more than *number* characters.
Action: Retry the command with fewer characters on the input line.

L1040 too many exported entries
Explanation: You have tried to export more than 65535 names.
Action: Retry with fewer names, creating an additional executable module if necessary.

L1041 resident-name table overflow
Explanation: The total length of all your resident-names, together with an overhead of 3 bytes for each name, is greater than the LINK386 limit. The internal LINK386 limit is 65,534 for LINK386 versions prior to 2.01.012 and is 1,048,576 for LINK386 version 2.01.012 and later versions.
Action: Reduce the number or the length of your resident names.

L1042 nonresident-name table overflow
Explanation: The total length of all your nonresident-names, together with an overhead of 3 bytes for each name, is greater than 65,534 for versions prior to 2.01.012 and is 1,048,576 for LINK386 version 2.01.012 and later versions.
Action: Reduce the number or the length of your nonresident-names.

L1043 relocation table overflow
Explanation: There are more than 65,535 load-time relocations for a single segment.
Action: Reduce the number of relocations in the source files and recompile or reassemble them. Interframe references generate load-time relocations.

L1044 imported-name table overflow
Explanation: The total length of all your imported-names, together with an overhead of 1 byte for each name, is greater than 65,535 bytes.
Action: Reduce the number or the length of your imported-names.

L1045 too many TYPDEF records
Explanation: An object module contains more than 255 TYPDEF records. These records describe communal variables. This error can only appear with programs produced by compilers that support communal variables.
Action: Reduce the number of TYPDEF records, breaking the module into smaller parts, if necessary.

L1046 too many external symbols in one module
Explanation: An object module specifies more than the limit of 1023 external symbols.
Action: Reduce the number of external symbols, breaking the module into smaller parts, if necessary.

L1047 too many group, segment, and class names in one module
Explanation: The program module contains too many group, segment, and class names.
Action: Reduce the number of groups, segments, or classes, and re-create the object files.

L1048 too many segments in one module
Explanation: An object module has more than 255 segments.
Action: Reduce the number of segments, splitting the module or combining some segments.

L1049 too many segments
Explanation: The program has more than the maximum number of segments. The /SEGMENTS option specifies the maximum allowed number; the maximum is 16375.
Action: Restart LINK386 using the /SEGMENTS option with an appropriate number of segments.

Fatal Error Messages (Part 2) 1050 - 1098

L1050 too many groups in one module
Explanation: LINK386 found more than 32 group definitions (GRPDEF) in a single module.
Action: Reduce the number of group definitions by splitting the module, by eliminating one or more group definitions, or combining group definitions.

L1051 too many groups
Explanation: The program defines more than 32 groups in addition to DGROUP.
Action: Reduce the number of group definitions by splitting the module, by eliminating one or more group definitions, or combining group definitions.

L1052 too many libraries
Explanation: You tried to link with more than 32 libraries.
Action: Combine libraries, or use modules that require fewer libraries.

L1053 out of memory for symbol table
Explanation: The program has more symbolic information, such as public, external, segment, group, class, and file names, than the amount that could fit in available real memory.
Action: Combine modules or segments and recreate the object files. Eliminate as many public symbols as possible or use shorter names.

L1054 requested segment limit too high
Explanation: There is not enough memory to allocate the necessary tables for the amount of segments requested.
Action: Reduce the number of segments by combining or creating additional executable modules.

L1057 data record too large
Explanation: A LEDATA record (in an object module) contained more than 1024 bytes of data. This is a translator (compiler or assembler) error.
Action: Note which translator (compiler or assembler) produced the incorrect object module and the circumstances, and contact your supplier.

L1060 program exceeds *number* bytes
Explanation: There is not enough memory to process all segments.
Action: Reduce the number of segments by combining or creating additional executable modules.

L1063 out of memory for debugging information
Explanation: LINK386 was given too many object files with debug information, and ran out of space to store them.
Action: Reduce the number of object files that have debug information.

L1064 out of memory - *name* heap exhausted
Explanation: The linker ran out of heap space; *name* = near or far.
Action: Reduce the number of background processes or install more memory.

L1070 *name* : segment size exceeds 64K
Explanation: A single segment contains more than 64K of code or data. This could be because you attempted to combine identically named segments.
Action: Try compiling (or assembling) and linking using a larger memory model or breaking up the named segment.

L1071 segment _TEXT larger than 65,520 bytes
Explanation: This error is likely to occur only in small-model C programs, but it can occur when any program with a segment named _TEXT is linked using the /DOSSEG option of the LINK386 command. Small-model C programs must reserve code addresses 0 and 1; the reserve is increased to 16 for alignment purposes.
Action: Make the program source code smaller, or change to a larger memory model.

L1072 common area longer than 65,536 bytes
Explanation: The program has more than 64K of communal variables. This error occurs only with programs produced by compilers that support communal variables.
Action: Rewrite your program using fewer or smaller communal variables.

L1073 file-segment limit exceeded
Explanation: There are more than 255 physical or file segments.
Action: Reduce the number of physical or file segments. You could use the Combine Contiguous Data (/PACKD) option for combining data segments or the Combine Contiguous Code (/PACKC) option for combining code segments.

L1074 *name* : group larger than 64K
Explanation: A group contains segments that total more than 65,536 bytes.
Action: Reduce the number or size of segments or remove segments from the group.

L1075 entry table larger than 65,535 bytes
Explanation: You have exceeded a linker table size limit because of an excessive number of entry names.
Action: Reduce the number of names in the modules that you are linking or create additional executable modules.

L1076 *name* : segment size exceeds *number* M
Explanation: The named segment is larger than the specified size.
Action: Break the segment into smaller segments and try again.

L1077 common area longer than 4G-1 bytes
Explanation: The space for the C languages common area is too big.
Action: If the load module is an .EXE, consider putting some routines in .DLL; otherwise, link without debugging information or create additional executable modules.

L1080 cannot open list file
Explanation: The disk or a directory is full, or an invalid file name was specified.
Action: Check that the file name specified is correct. Delete or move files to make space and restart LINK386.

L1081 out of space for run file
Explanation: The disk on which the .EXE file is being written is full.
Action: Delete or move files to make space and restart LINK386.

L1082 *name* : stub file not found
Explanation: The stub file specified in the module definition file could not be found.
Action: Check that the correct path to the stub file has been specified.

L1083 cannot open run file - *reason*
Explanation: The run file could not be opened for the stated reason.
Action: Correct the problem and restart LINK386.

L1088 out of space for list file
Explanation: The disk on which the listing file is being written is full.
Action: Delete or move files to make space and restart LINK386.

L1089 *filename* : cannot open response file
Explanation: LINK386 cannot find the specified response file. This usually indicates a typing error.
Action: Include the drive specifier or path, or both, for the response file.

L1091 unexpected end-of-file on library
Explanation: The disk containing the library has probably been removed or is corrupted.
Action: Replace the disk containing the library and restart LINK386.

L1092 cannot open module definition file
Explanation: The specified module definition file cannot be opened, or an invalid file name was specified.
Action: Check that the specified file name is correct. Include the drive specifier or path, or both, for the module definition file.

L1093 *name* : object not found
Explanation: LINK386 could not open the object module you specified.
Action: Specify full path name or directory in which object module resides.

L1096 unexpected end-of-file
Explanation: LINK386 encountered an end-of-file character while reading an input file AND expected more information.
Action: Check input files for errors and relink.

L1097 I/O error - *string*
Explanation: The linker encountered the I/O error shown while reading from a file.
Action: Make sure the file is not corrupted or on a bad disk sector and relink.

L1098 cannot open include file *filename* - *reason*
Explanation: LINK386 could not open the include file for the stated reason.
Action: Correct the problem and restart LINK386.

Fatal Error Messages (Part 3) 1100 - 1130

L1100 stub .EXE file invalid
Explanation: The stub file specified in the module definition file is not a valid .EXE file.
Action: Ensure that the stub file is an executable file.

L1101 invalid object module
Explanation: One of the object modules was incorrectly formed during compilation or assembly.
Action: Recompile or reassemble your source code. If the error persists, contact your supplier.

L1102 unexpected end-of-file
Explanation: An invalid format for a library was found.
Action: Restore the library file from your backup disk and restart LINK386. If this does not work, rebuild your library or contact your supplier.

L1103 *name* : attempt to access data outside segment bounds
Explanation: A data record in an object module specified data extending beyond the end of a segment. This is a translator error.
Action: Note which translator (compiler or assembler) produced the incorrect object module and the circumstances, and contact your supplier.

L1104 *filename* : not valid library
Explanation: The specified file is not a valid library file. This error causes LINK386 to stop running.
Action: Ensure that the named file is a valid library file and restart LINK386. If this does not work, rebuild your library or contact your supplier.

L1105 invalid object due to aborted incremental compile
Explanation: An object file from an aborted compile is trying to be linked.
Action: Recompile the source that produced the bad compile and then relink.

L1106 unknown COMDAT allocation type for *name* ; record ignored
Explanation: The COMDAT (record in .OBJ) allocation type for the named COMDAT was not valid.
Action: Recompile or reassemble .OBJ and try again. If that does not work, contact your supplier.

L1107 unknown COMDAT selection type for *name* ; record ignored
Explanation: The COMDAT (record in .OBJ) selection type for the named COMDAT is not valid.
Action: Recompile or reassemble .OBJ and try again. If that does not work, contact your supplier.

L1108 invalid format of debugging information
Explanation: An error was detected in a segment that contains debug information.
Action: Recompile or reassemble and try again. If that does not work, try relinking without using the /DE or /CO options.

L1113 unresolved COMDEF; internal error
Explanation: LINK386 encountered a COMDEF (record in .OBJ) in pass 2 that was not defined in pass 1.
Action: Recompile or reassemble and try again. If that does not work, contact your supplier.

L1114 unresolved COMDAT *name* : internal error
Explanation: LINK386 found specified COMDAT in pass 2 that does not correspond to COMDATs found in pass 1.
Action: Recompile or reassemble and try again. If that does not work, contact your supplier.

L1117 unallocated COMDAT *name* ; internal error
Explanation: The linker encountered COMDAT in pass 2 for which space was not allocated during pass 1.
Action: Recompile or reassemble and try again. If that does not work, contact your supplier.

L1121 *name* : group larger than 4G-1 bytes
Explanation: The group indicated is too large.
Action: Recompile or reassemble and try again. Remove segments from the group or create additional executables if necessary.

L1123 *name* : segment defined both 16- and 32-bit
Explanation: The segment named was defined as both 16-bit and 32-bit.
Action: Create two segments (one for 16-bit, one for 32-bit), or make sure the segment is defined one way and relink.

L1126 conflicting IOPL-parameter-words value

Explanation: The IOPL parameter words in the .DEF file does not exactly match those in the corresponding EXPDEF object record.
Action: Make sure .DEF file coincides with that defined in .OBJ. Relink.

L1128 too many nested include files in module-definition file
Explanation: The .DEF file exceeded a nesting level.
Action: Combine some nestings into one .DEF file and try again.

L1129 missing or bad include file name
Explanation: LINK386 could not find a file included by .DEF file.
Action: Make sure the file exists and can be located and that any path is specified correctly. Try again.

L1130 internal fix-up applied to undefined area at *offset* in object *number*
Explanation: LINK386 attempted to apply an internal fix-up beyond the defined limits of the object.
Action: This is probably a compiler or assembler error. Revise the source file and re-create the object file. If that does not work, contact your supplier.

Fatal Error Messages (Part 4) 1200 - 1218

L1200 out of memory for page range information
Explanation: The total size of all page range information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1201 out of memory for contribution information
Explanation: The total size of the contribution information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1202 out of memory for executable string information
Explanation: The total size of all executable string information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1203 out of memory for identifier mangler DLL information
Explanation: The total size of all identifier mangler DLL information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1204 out of memory for object page table information
Explanation: The total size of the object page table information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1205 out of memory for fixup record information
Explanation: The total size of all fixup record information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1206 out of memory for fix up page table information
Explanation: The total size of all fix up page table information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1207 out of memory for import module information
Explanation: The total size of the import module information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1208 out of memory for page directory information
Explanation: The total size of the page directory is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1209 out of memory for object page directory information

Explanation: The total size of the object page director is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1210 out of memory for module reference table information
Explanation: The total size of the module reference table is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1211 out of memory for chained relocation hash buckets
Explanation: The total size of the relocation bucket is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1212 out of memory for chained relocation hash tables
Explanation: The total size of the relocation hash tables is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1213 out of memory for entry point information
Explanation: The total size of the entry point information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1214 out of memory for entry table area
Explanation: The total size of the entry table are is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1215 out of memory for imported names information
Explanation: The total size of the imported names information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1216 out of memory for comdat iterated data
Explanation: The total size of all comdat iterated is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1217 out of memory for back patch information
Explanation: The total size of the back patch information is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

L1218 out of memory for property list tables
Explanation: The total size of the property list table is greater than the internal limit in LINK386.
Action: Reduce the size of the executable, consider using dynamic link libraries to split the program into smaller modules.

Nonfatal Error Messages 2000 - 2063

L2000 imported starting address
Explanation: A MODEND, or starting address record, referred to an imported name. Imported program-starting addresses are not supported.
Action: The starting address record must refer to a non imported name. Edit the source file and recompile or reassemble.

L2002 fix-up overflow at *location* in segment *name*
Explanation: A fix up overflow occurred near *location*, in the named segment. See Conditions That Can Cause LINK386 Error 2002.
Action: Revise the source file and re-create the object file. If that does not work, contact your supplier.

L2003 inter segment self-relative fix-up at *location* in segment *name*
Explanation: LINK386 detected an inter segment self-relative fix-up. A self-relative fix-up cannot be in another segment. This is probably caused by a near reference to procedure located in another segment.

Action: Revise the source file and re-create the object file. If that does not work, contact your supplier.

L2005 fix-up type unsupported at *location* in segment *name*
Explanation: LINK386 detected an unsupported fix up type. This is probably a compiler or assembler error.
Action: Revise the source file and re-create the object file. If that does not work, contact your supplier.

L2010 too many fix-ups in LIDATA record
Explanation: There are more fix ups applying to a LIDATA record than will fit into the LINK386's 1024 byte buffer. The buffer is divided between the data in the LIDATA record itself and the run-time relocation items. These are 8 bytes each, so the maximum varies from 0 to 128. This is probably a compiler error.
Action: Revise the source file and re-create the object file. If that does not work, contact your supplier.

L2011 *name* : NEAR/HUGE conflict
Explanation: There are conflicting NEAR and HUGE attributes for a communal variable. This error can occur only with programs produced by compilers that support communal variables.
Action: Specify only one of these attributes.

L2012 *name* : array-element size mismatch
Explanation: A far communal array has been declared with two or more different array-element sizes (for example, an array declared once as an array of characters and once as an array of real numbers). This error occurs only when using compilers that support far communal arrays.
Action: Match the definitions and re-create the object module or modules.

L2013 LIDATA record too large
Explanation: A LIDATA record in an object module contains more than 512 bytes of data. It is likely that one of your assembler modules contains a complex structure definition or a series of deeply-nested DUP operators. (LIDATA is a DOS term.)
Example: The following structure definition causes this error:

```
alpha   DB     10DUP(11 DUP(12 DUP(13 DUP(...))))
```

Action: Simplify the structure definition and reassemble the module.

L2022 *name* (alias *internalname*) : export undefined
Explanation: A name has been directed to be exported but is not defined anywhere.
Action: Edit the source file and define the export.

L2023 *name* (alias *internalname*) : export imported
Explanation: An imported name has been directed to be exported. Items that are not in the source file itself cannot be exported. You cannot export this imported name.
Action: Edit the source file to not export the imported name.

L2024 *name* : special symbol already defined
Explanation: Your program defined a symbol name that LINK386 already used for one of its low-level symbols. For example, the linker generates special names for overlay support.
Action: Edit the source file and choose another name for the symbol.

L2025 *name* : symbol defined more than once
Explanation: A symbol has been defined more than once in the object file.
Action: Edit the source file, removing the extra symbol definition.

L2026 entry ordinal *number, name name* : multiple definitions for same ordinal
Explanation: More than one entry point name has been assigned to the same ordinal in the module definition file.
Action: Edit the module definition file to correct the usage of the ordinal.

L2027 *number* : ordinal too large for export
Explanation: You tried to export more than 3072 names or indicated too large of an ordinal.
Action: Edit the source file or define smaller ordinals in the module definition file.

L2029 *string* : unresolved external
Explanation: A symbol declared to be external in one or more modules was not found among the given objects and libraries.
Action: Supply files that will resolve these external calls.

L2030 starting address not code (use class 'CODE')
Explanation: You specified a starting address to LINK386 that is not within a segment of type 'CODE.'
Action: Reclassify the segment to CODE, or correct the starting point.

L2044 *string* : symbol multiple defines, use /NOE
Explanation: The symbol shown was defined more than once, perhaps for different things.
Action: Recompile with the /NOEXDICTIONARY option.

L2047 IOPL attribute conflict - segment: *name* in group:*name*
Explanation: The segment indicated within the group shown has different IOPL attributes from the rest of the segments in the group.
Action: Remove the segment from the group or make sure all segments have the same IOPL attributes. Relink.

L2050 use16/use32 attribute conflict - segment:*name* in group:*name*
Explanation: The segment indicated within the group shown has a different USE16/USE32 attribute from the rest of the segments in the group.
Action: Remove the segment from the group or make sure all segments have the same USE16/USE32 attribute.

L2052 *name* : unresolved external - possible calling convention mismatch
Explanation: LINK386 encountered an undefined external, which could be a fast-call/C-call mismatch.
Action: Make sure the external is defined and called the same way (for example, all C-calls).

L2053 call gates are NOT allowed in 32-bit object if its size exceeds 64K - memory object number *number*
Explanation: The memory object indicated is larger than 64K and uses call gates.
Action: Remove the call gates or break the object into smaller portions. Relink.

L2054 data for invalid page in segment *name*
Explanation: LINK386 encountered data past the defined end of initialized data in the segment indicated.
Action: The .OBJ is probably corrupted. Recompile or reassemble and try again. If that does not work, contact your supplier.

L2055 fix-up for invalid page at *location* in segment *name*
Explanation: LINK386 encountered a fixup past the defined end of initialized data in the segment indicated.
Action: The .OBJ is probably corrupted. Recompile or reassemble and try again. If that does not work, contact your

L2056 object type conflict - segment: *name* in group: *name*
Explanation: The segment indicated within the group shown has a different TYPE attribute from the rest of the segments in the group.
Action: Remove the segment from the group or make sure all segments have the same TYPE attribute.

L2057 duplicate of *name* with different size found; record ignored
Explanation: LINK386 encountered two COMDAT records with the selection type "SAME SIZE." The records have the same name but have different sizes.
Action: The .OBJ is probably corrupted. Recompile or reassemble and try again. If that does not work, contact your

L2058 different duplicate of *name* found; record ignored
Explanation: LINK386 encountered two COMDAT records with the selection type "EXACT." The records have the same name but have different sizes.
Action: The .OBJ is probably corrupted. Recompile or reassemble and try again. If that does not work, contact your

L2059 size of the data block associated with *name* exceeds 4G
Explanation: LINK386 encountered a continuation COMDAT record whose additional size made the communal data too large.
Action: The .OBJ is probably corrupted. Recompile or reassemble and try again. If that does not work, contact your

L2061 no space for the data block associated with *record name* inside the *segment name*
Explanation: While being allocated space for a COMDAT record inside the segment indicated, the segment grew larger than 64K.
Action: Make the segment smaller or move communal data into a different segment.

L2062 continuation of COMDAT *name* has conflicting attributes; record ignored
Explanation: While concatenating the COMDAT record indicated, LINK386 found attributes that differ from those defined in the initial COMDAT record.
Action: The .OBJ is probably corrupted. Recompile or reassemble and try again. If that does not work, contact your

L2063 *name* allocated in undefined segment
Explanation: LINK386 encountered a COMDAT record in an undefined explicit allocation segment.
Action: The .OBJ is probably corrupted. Recompile or reassemble and try again. If that does not work, contact your supplier.

Warning Error Messages 4000 - 4087

L4000 seg disp. included near *location* in segment *name*

Explanation: This error is caused by using the LINK386 Warning of Fix-ups (/W) option.
Action: The segment name and the location offset is displayed.

L4001 frame-relative fix-up, frame ignored near *location* in segment *name*
Explanation: A fix up occurred with a frame segment different from the target segment where either the frame or the target segment is not absolute. Such a fix-up is meaningless in the OS/2 environment, so the target segment is assumed for the frame segment. This error sometimes occurs if a 32-bit data item is referenced as if it were in a 16-bit segment or a 16-bit segment referenced as if it were in a 32-bit segment.
Action: Check that this is acceptable.

L4002 frame-relative absolute fix-up near *location* in segment *name*
Explanation: A fix up occurred with a frame segment different from the target segment where both frame and target segments are absolute. This fix up is processed using base-offset arithmetic, but the warning is issued because the fix up might not be valid in the OS/2 environment. This error sometimes occurs if a 32-bit data item is referenced as if it were in a 16-bit segment or a 16-bit segment referenced as if it were in a 32-bit segment.
Action: Check that this is acceptable.

L4003 intersegment self-relative fix up at *location* in segment *name*
Explanation: LINK386 found an intersegment self-relative fix-up at the specified location. This might cause a problem with the executable file.
Action: The error might have been caused by the way the program was written or when it was compiled or assembled.

L4004 possible fix-up overflow at *location* in segment *name*
Explanation: LINK386 found a possible fix-up overflow at the specified location. This might cause a problem with the executable file.
Action: The error might have been caused by the way the program was written or when it was compiled or assembled.

L4005 32-bit fix-up in 16-bit record ignored at *location* in segment *name*
Explanation: LINK386 encountered a 32-bit fix-up in a 16-bit record at the specified location. This might cause a problem with the executable file.
Action: The error might have been caused by the way the program was written or when it was compiled or assembled.

L4006 illegal 16-bit flat-relative offset fix-up at *location* in object *name*
Explanation: LINK386 encountered an illegal 16-bit flat relative offset fix-up at the specified location. This might cause a problem with the executable file. This error sometimes occurs if a 32-bit data item is referenced as if it were in a 16-bit segment or a 16-bit segment referenced as if it were in a 32-bit segment.
Action: The error might have been caused by the way the program was written or when it was compiled or assembled.

L4007 illegal 16-bit flat-relative pointer fix-up at *location* in object *name*
Explanation: LINK386 encountered an illegal 16-bit flat-relative pointer fix-up at the specified location. This error sometimes occurs if a 32-bit data item is referenced as if it were in a 16-bit segment or a 16-bit segment referenced as if it were in a 32-bit segment.
Action: The error might have been caused by the way the program was written or when it was compiled or assembled.

L4008 aliased fix-up to non-alias object near *location* in object *name*
Explanation: LINK386 encountered an aliased fix-up to a non-alias object at the specified location. This error sometimes occurs if a 32-bit data item is referenced as if it were in a 16-bit segment or a 16-bit segment referenced as if it were in a 32-bit segment.
Action: The error might have been caused by the way the program was written or when it was compiled or assembled.

L4009 illegal target of flat-relative fix-up ignored at *number* in segment *name*
Explanation: LINK386 encountered an illegal flat-relative fix-up at the specified location.
Action: The error might have been caused by the way the program was written or when it was compiled or assembled.

L4010 invalid alignment specification; assuming *number*
Explanation: The number following the /ALIGNMENT option is not a power of 2, or is not in numerical form. The maximum alignment value is 4096.
Action: If the default alignment of 512 is not acceptable, restart LINK386 using a valid number.

L4017 *name* : unrecognized option name; option ignored
Explanation: The option specified is not valid for LINK386.
Action: Specify a valid option or remove the unrecognized option and relink.

L4018 missing or bad application type; option *name* ignored
Explanation: The /PMTYPE option was specified without an application type or with an invalid application type.
Action: Relink with an application type of PM, VIO, or NOVIO.

L4020 *name* : code segment size exceeds 65,500
Explanation: The code segment indicated is larger than 65,500 bytes and might not be reliable.
Action: Break the segment into smaller segments and try again.

L4021 no stack segment
Explanation: The program does not contain a stack segment defined with the STACK combine type. Normally, every .EXE program should have a stack segment with the combine type specified as STACK.
Action: You can ignore this message if you have a specific reason for not defining a stack or for defining one without the STACK combine type.

L4022 *name1, name2* : groups overlap
Explanation: Two groups are defined in such a way that one starts in the middle of another. This can occur if you defined segments in a module definition file or assembly file and did not correctly order the segments by class.
Action: Edit the source file and reorder the segments in the group.

L4023 *name* (alias): export internal name conflict
Explanation: An exported name, or its associated internal name, conflicts with an already defined public symbol.
Action: Edit the source file using new names.

L4024 *name* : multiple definitions for export name
Explanation: The module named has been exported more than once with different internal names. All internal names except the first one are ignored.
Action: Edit the source file using new names.

L4025 *modname impname (intname)* : import internal name conflict
Explanation: An imported name, or its associated internal name, is also defined as an exported name. The import name is ignored. The conflict could come from a definition in either the module definition file or an object file.
Action: Edit the source file or module definition file using new names.

L4026 *modname impname (intname)* : self-imported
Explanation: The module definition file directed that a name be imported from the module being produced.
Action: Edit the module definition file.

L4027 *name* : multiple definitions for import internal name
Explanation: An imported name, or its associated internal name, is imported more than once. The imported name is ignored after the first mention.
Action: Check that the name has been defined correctly.

L4028 *name* : segment already defined
Explanation: A segment is defined more than once with the same name in the module definition file. Segments must have unique names for LINK386. All definitions with the same name are ignored after the first mention.
Action: Check that the segment has been defined correctly.

L4029 *name* : DGROUP segment converted to type data
Explanation: A segment that is a member of DGROUP has been defined as type CODE in a module definition file or object file. This probably happened because a CLASS keyword in a SEGMENTS statement was not given.
Action: Check the module definition file syntax.

L4030 *name* : segment attributes changed to conform with automatic data segment
Explanation: The segment named is defined in DGROUP, but the *shared* attribute is in conflict with the *instance* attribute.
Example: The *shared* attribute is NONSHARED and the *instance* attribute is SINGLE, or the *shared* attribute is SHARED and the *instance* attribute is MULTIPLE. The bad segment is forced to have the right *shared* attribute and the link continues.
Action: Check that the LINK386 action is acceptable.

L4031 *name* : segment declared in more than one group
Explanation: A segment is declared to be a member of two different groups.
Action: Correct the source file and re-create the object files.

L4032 *name* : code-group size exceeds 65500 bytes
Explanation: The code group indicated is larger than 65500 bytes and therefore might not be reliable.
Action: Break the group into smaller groups or remove one or more segments and try again.

L4036 no automatic data segment
Explanation: The program or dynamic link library did not define a group named DGROUP. This is recognized by LINK386 as the automatic data segment.
Action: Edit the source file.

L4038 program has no starting address
Explanation: The program did not contain a starting address. Physical Device Drivers do not have program starting addresses, so this error can be ignored when linking Physical Device Drives.

Action: Recompile the program and try again.

L4044 CODE segment :*name* in DATA group:*name* ; assuming DATA
Explanation: A CODE statement in a module definition file was used to define default attributes for a DATA segment.
Action: Define a CODE statement and relink.

L4045 name of output file is *name*
Explanation: A dynamic link library file was created without specifying an extension. In such cases, LINK386 supplies an extension of .DLL. This is to warn you in case you expected an .EXE file to be generated
Action: No action.

L4046 DATA segment: *name* in CODE group: *name* ; assuming CODE
Explanation: A DATA statement in a module definition file was used to define default attributes for a CODE segment.
Action: Define a DATA statement and relink.

L4048 ignoring non-zero heap size
Explanation: The module definition file does not contain a HEAPSIZE statement.
Action: Edit the file and relink.

L4049 ignoring non-zero stack size
Explanation: The module definition file does not contain a STACKSIZE statement.
Action: Edit the file and relink.

L4051 *filename* : cannot find library
Explanation: LINK386 could not find the specified library file.
Action: Enter a new file name, a new path specification, or both.

L4053 VM.TMP :illegal file name; ignored
Explanation: VM.TMP cannot be used for an object file name.
Action: Rename the file and restart LINK386.

L4054 *filename* : cannot find file
Explanation: LINK386 could not find the specified file.
Action: Enter a new file name, a new path specification, or both.

L4067 changing default resolution for weak external *name* from *oldname* to *newname*
Explanation: LINK386 encountered a redefinition of a default resolution and is changing it to the value indicated.
Action: If the change is OK, no action is required; otherwise, fix the module definition file and try again.

L4068 ignoring stack size greater than 64K
Explanation: LINK386 encountered a stack greater than 64K or zero and is assuming a stack size of 65,534.
Action: Edit the file and relink.

L4069 filename truncated to *name*
Explanation: LINK386 encountered a file name greater than 256 bytes (including terminating null) and truncated it to the size indicated.
Action: Edit the file and relink.

L4071 application type not specified; assuming *name*
Explanation: An application type of WINDOWAPI, WINDOWCOMPAT, NOTWINDOWCOMPAT, or PRIVATE was not specified. LINK386 is assuming the application type indicated.
Action: Edit the file and relink.

L4072 changing application type from *oldname* to *newname*
Explanation: The application type specified with /PMTYPE is different from that in .DEF file. LINK386 is using the application type indicated.
Action: Edit the file and relink.

L4073 *name* : 32-bit aliased data segment size exceeds 64K
Explanation: The segment indicated is greater than 64K in length and is a 32-bit aliased data segment
Action: If this is expected, do nothing; if not, break into smaller segments.

L4074 attribute conflict for segment *name* ; ignoring attribute *type*
Explanation: The segment indicated for the .DEF file is defined with conflicting characteristics. LINK386 is ignoring the attribute indicated.
Action: Edit the file and relink.

L4075 object type conflict - assuming *name*
Explanation: The .DEF files specified conflicting attributes for an object; only one of the following attributes are allowed: RESIDENT, NONPERMANENT, PERMANENT, CONTIGUOUS, or DYNAMIC. LINK386 is assuming the attribute indicated.

Action: Edit the file and relink.

L4077 symbol *name* not defined; ordered allocation ignored
Explanation: While doing ordered allocation of COMDAT records, LINK386 encountered an undefined COMDAT record; ordered allocation is determined from the .DEF file.
Action: Edit the file and relink.

L4079 symbol *name* already defined for ordered allocation; duplicate ignored
Explanation: While processing ORDER list in .DEF file, LINK386 encountered a COMDAT record already defined for ordered allocation.
Action: Edit the file and relink.

L4080 changing substitute name for alias *name* from *name* to *name*
Explanation: LINK386 encountered an alias redefiniton and is changing it to the values indicated.
Action: If this is OK, no action is required. Otherwise, edit the file and relink.

L4082 *name* ignored for module with 16-bit starting address
Explanation: LINK386 encountered a DLL module with a 16-bit entry point requesting termination. Only modules with 32-bit entry points can specify DLL termination.
Action: Remove the termination request from the module definition file.

L4083 invalid base address specification; assuming *number*
Explanation: The base address specified with the /BASE option or in the module definition file is illegal, and LINK386 is assuming the given value.
Action: Change the base address if necessary; otherwise, ignore the message.

L4084 module name truncated to *string*
Explanation: The module name was truncated to the number of characters indicated.
Action: If this action is satisfactory, no action is required. Otherwise, edit the module definition file and shorten the name.

L4085 *name* (alias *alias name*): forwarder entry created for imported export
Explanation: LINK386 created a forwarder entry within the entry table for the named export.
Action: If this action is satisfactory, no action is required.

L4087 internal fix-up applied to uninitialized area at *offset* in object *number*
Explanation: LINK386 attempted to apply an internal fix-up beyond the initialize limits of the object.
Action: If this is acceptable, no action is required. If the problem continues, you might want to disable based addressing.

L4090 cannot load identifier manipulation DLL *name*
Explanation: LINK386 detected an error while trying to load an identifier manipulator dynamic link library. This DLL was specified in an object file, and is used by LINK386 to demangle a compiler generated mangled name into a function prototype when printing an error message. Error messages will not be demangled for this object file.
Action: Make sure the appropriate identifier manipulator DLL is in the LIBPATH.

L4091 cannot locate *procedure* in identifier manipulation DLL *name*
Explanation: LINK386 detected an error while trying to load a procedure from an identifier manipulator dynamic link library. This DLL was specified in an object file, and is used by LINK386 to demangle a compiler generated mangled name into a function prototype when printing an error message. Error messages will not be demangled for this object file.
Action: Make sure the appropriate identifier manipulator DLL is in the LIBPATH.

L4092 too many identifier manipulation DLLs
Explanation: Too many identifier manipulator dynamic link libraries have been specified. These DLLs are specified in object files, and are used by LINK386 to demangle compiler generated mangled names into function prototypes when printing an error messages. Error messages may not be demangled for some object files.
Action: Reduce the number of different compilers used to create the objects.

L4093 cannot initialize identifier manipulation DLL *name*
Explanation: LINK386 detected an error while trying to initialize an identifier manipulator dynamic link library. This DLL was specified in an object file, and is used by LINK386 to demangle a compiler generated mangled name into a function prototype when printing an error message. Error messages will not be demangled for this object file.
Action: Make sure the appropriate identifier manipulator DLL is in the LIBPATH.

L4094 increasing stack size from *number* to *number*
Explanation: The stack size specified by either

1. size of a segment with combine type stack
2. STACKSIZE statement in the .DEF file
3. /STACK LINK386 command line option

will cause a system error if the program is executed on an OS/2 2.x system. LINK386 has changed the stacksize to a

larger value to preserve compatibility.
Action: No action required, LINK386 has corrected the problem. To eliminate the warning message, restart LINK386 and specify the new stack size.

Conditions That Can Cause LINK386 Error 2002

LINK386 Error 2002 can be caused by the following conditions:

- A group is larger than 64K.
- The program contains an intersegment short jump or intersegment short call.
- The name of a data item in the program conflicts with that of a subroutine in a library included in the link.
- An EXTRN declaration in an assembler language source file appeared inside the body of a segment, as in the following example:

```
code    SEGMENT public  'CODE'
        EXTRN   main:far
start   PROC    far
        call    main
        ret
start   ENDP
code    ENDS
```

The following construction is preferred:

```
        EXTRN   main:far
code    SEGMENT public  'CODE'
start   PROC    far
        call    main
        ret
start   ENDP
code    ENDS
```

Make Message File (MKMSGF) Error Messages

MKMSGF: Codepage %s is all zeroes
Explanation: The code-page ID specified with the /P option is zero. The message file is built with a code-page of zero.
Action: Retry the command using the correct code-page specification.

MKMSGF: Codepage %s error in numeric conversion
Explanation: The code-page ID specified with the /P option is not numeric. The message file is built with a code-page of zero.
Action: Retry the command using the correct code-page specification.

MKMSGF: Codepage %s is too large
Explanation: The code-page ID specified with the /P option is too large. The message file is built with a code-page of zero.
Action: Retry the command using the correct code-page specification.

MKMSGF: Country %u is not supported
Explanation: The country ID specified within the /D option is not supported. MKMSGF processing is stopped.
Action: Retry the command using the correct country code specification.

MKMSGF: DBCS code page not found
Explanation: No DBCS code page has been found that supports the DBCS range specified in the /D option. MKMSGF processing is stopped.
Action: Retry the command using the correct DBCS ranges or country ID for the input message file.

MKMSGF: Error reading input file

Explanation: Error during input from source file.
Action: Make sure the source message file exists and that the drive is ready. Retry the command.

MKMSGF: Error writing output file

Explanation: Error during output to target file.
Action: Make sure there is sufficient disk space or that the drive is ready. Retry the command.

MKMSGF: File not found

Explanation: Input file could not be found.
Action: Retry the command, using the correct source message file name.

MKMSGF: Infile[.ext] outfile[.ext] [/V]
[/D <DBCS range or country>] [/P <code page>] [/L <language id,sub id>]

Explanation: This is the proper syntax for MKMSGF. It is displayed when no operands are specified on the command line and after some syntax errors.
Action: None.

MKMSGF: Input file same as output file

Explanation: The input and output file names are the same. Processing is stopped.
Action: Correct the command line or the control file and restart MKMSGF.

MKMSGF: Insufficient storage

Explanation: Not enough storage to execute program or too many messages in the file. Message limit is about 6000.
Action: Reduce the number of programs running in your system, or reduce the size of the message file by either deleting messages or by reducing the size of each message. Retry the command.

MKMSGF: Invalid language or sub id

Explanation: The language family ID specified with the /L option is not supported.
Action: Retry the command using the correct language family ID.

MKMSGF: Invalid message file format

Explanation: Input file is not a recognizable message text file.
Action: If an incorrect file name was entered, retry the command with the correct source message file name.

MKMSGF: Language family %s is all zeroes

Explanation: The language family ID specified with the /L option is zero. The message file is built with a language family ID of zero.
Action: Retry the command using the correct language family ID.

MKMSGF: Language family %s error in numeric conversion

Explanation: The language family ID specified with the /L option is not numeric. The message file is built with a language family ID of zero.
Action: Retry the command using the correct language family ID.

MKMSGF: Language family %s is too large

Explanation: The language family ID specified with the /L option is not supported. The message file is built with a language family ID of zero.
Action: Retry the command using the correct language family ID.

MKMSGF: Message ID out of sequence

Explanation: A message was detected that was out of the required sequential order.
Action: Correct the error by editing your source message file and renumbering the messages. You might also want to delete or insert the appropriate message numbers to achieve the required sequential order.

MKMSGF: Message XXXX too long

Explanation: The message was too long to be processed (limit is approximately 2K characters).
Action: Correct the error by editing your source message file and making the message shorter. Then, retry the command.

MKMSGF: More than NN codepages entered

Explanation: A maximum of NN code-page IDs can be specified for a single message file. Only the first NN is accepted.ECE
Action: Retry the command using the correct code-page specifications.

MKMSGF: No sub id using 1 default

Explanation: The language version ID specified with the /L option is either invalid or not supported. The message file is built using the default value shown.
Action: Retry the command using the correct language version ID.

MKMSGF: Sub id %s error in numeric conversion

Explanation: The language version specified with the /L option is not numeric. The message file is built with a

default language version.
Action: Retry the command using the correct language version ID.

MKMSGF: Syntax error
Explanation: You entered the command incorrectly.
Action: Retry the command using proper syntax. To display the proper syntax, type MKMSGF at the command line.

Message Segment Binder (MSGBIND) Error Messages

MSGBIND: I/O error seeking infile
Explanation: A disk error occurred while seeking either the message file or the .EXE file.
Action: Run CHKDSK on the drive containing the file and restart MSGBIND.

MSGBIND: I/O error writing file
Explanation: A disk error occurred while writing messages to the .EXE file.
Action: Run CHKDSK on the drive containing the .EXE file, and restart MSGBIND.

MSGBIND: Must specify .EXE file before message file
Explanation: The input file was in error.
Action: Correct input file.

MSGBIND: Must specify message file before message number
Explanation: The input file was in error.
Action: Correct the input file.

MSGBIND: Out of memory, needed *xxxx* bytes
Explanation: There was not enough memory available to run MSGBIND.
Action: Reduce the number of programs presently running in your system and restart MSGBIND.

MSGBIND: Premature EOF during copy
Explanation: The .EXE file was not built correctly.
Action: Rebuild the .EXE file and restart MSGBIND.

MSGBIND: Unable to create temp file-MSGBIND.TMP
Explanation: An error occurred while creating the intermediary file MSGBIND.TMP.
Action: Delete or move files to make disk space available. If MSGBIND.TMP is present as a read-only file, it must first be deleted. Restart MSGBIND.

MSGBIND: Unable to open *xxxxxxxx*
Explanation: The input file specified was not found or an error occurred when opening the message file.
Action: Restart MSGBIND using the correct input file name or a backup copy of the message file.

Number of Message Files exceeded. Only nnn allowed
Explanation: The number of message files specified in the input file exceeds the maximum number of files that MSGBIND can process at one time.
Action: Correct the input file or combine message files, and then restart MSGBIND.

Reading messages from *xxxxxxxx*
Explanation: Messages from the displayed message file are being read into memory.
Action: None. This message is for information only.

Reading messages from *xxxxxxxx* -file not found
Explanation: The message file was not found in the path specified in the input file.
Action: Edit the input file, correcting the path or file name, or both. Restart MSGBIND.

Reading messages from *xxxxxxxx* -not created with MKMSGF program
Explanation: The message file displayed was not created using the MKMSGF program.
Action: Convert the source message file to a formatted message file using the MKMSGF program, and restart MSGBIND. See the Make Message File (MKMSGF).

Reading messages from *xxxxxxxx* -not enough memory
Explanation: There was not enough memory available to store the messages.
Action: Reduce the number of programs running in your system, and restart MSGBIND.

The object that would contain the bound messages would be too large. Refer to the Toolkit Documentation for more information.

Explanation: (self explanatory)
Action: Do one or more, of the following:

1. Modify the module definition file to isolate the message object, and then relink the application.
2. Reduce the number of messages to be bound to the application as specified in the input file.
3. Reduce the size of the messages and rebuild the message file with MKMSGF.
4. Then restart MSGBIND.

Unable to bind message nnnn, message segment would exceed 64K. The remaining messages will NOT be bound.

Explanation: (self explanatory)
Action: Do one or more, of the following:

1. Modify the module definition file to isolate the message object, and then relink the application.
2. Reduce the number of messages to be bound to the application as specified in the input file.
3. Reduce the size of the messages and rebuild the message file with MKMSGF.
4. Then restart MSGBIND.

Unable to locate call to Dos32GetMessage, messages will not be bound.

Explanation: The executable file to be modified makes no function calls to the message retriever, or was not linked with the correct library file.
Action: If the application does use the message retriever function, such as DosGetMessage, relink the application using the correct library file, and then restart MSGBIND.

If not, messages cannot be bound to the application.

Updating *xxxxxxxx*

Explanation: The .EXE file name displayed is being updated with the requested messages.
Action: None. This message is for information only.

Updating *xxxxxxxx* -file not found

Explanation: The .EXE file name was not found in the path specified in the input file.
Action: Edit the input file, correcting the path or file name, or both. Restart MSGBIND.

Updating *xxxxxxxx* -not linked with MSGSEG.OBJ

Explanation: The .EXE file name displayed made no DosGetMessage functions, or the .EXE file was not linked with the correct library. Messages can only be bound to applications that call message retriever functions, such as DosGetMessage.
Action: If the .EXE file does make calls to message retriever functions, relink the application using the correct library, and restart MSGBIND.

usage: MSGBIND scriptfile

Explanation: The command was entered incorrectly.
Action: Restart MSGBIND. Refer to Syntax for the correct syntax.

Warning: No msgseg data found in new executable

Explanation: The executable file to be modified makes no function calls to the message retriever, or was not linked with the correct library file.
Action: If the application does use the message retriever function, such as DosGetMessage, relink the application using the correct library file, and then restart MSGBIND.

If not, messages cannot be bound to the application.

WARNING: Skipping messages for this file

Explanation: The .EXE file was in error, so the messages were not bound to it. Either the .EXE file does not exist, or it has not been linked with the correct library.
Action: If the .EXE file name is correct, relink the application using the correct library, and restart MSGBIND.

WARNING: Skipping message numbers for this file

Explanation: The message file was in error, so all messages from this message file were ignored. The message file might not exist, might not be formatted correctly, or there might not be enough memory to store all the messages.
Action: If the message file name is correct and has been correctly formatted, check the memory available for the file. Restart MSGBIND.

WARNING: *xxxx* is an invalid message number

Explanation: The message number specified was not found in the message file.
Action: Edit the input file and correct any message numbers that are in error. Restart MSGBIND.

Writing messages

Explanation: The .EXE file is being updated with the messages requested.
Action: None. This message is for information only.

Resource Compiler Error Messages

The error, warning, and information messages produced by the Resource Compiler (RC) are listed below. The message number indicates the severity of the error:

Message Number	Severity
1000 - 1999	Severe Errors
2000 - 2999	Error Messages
3000 - 3999	Warning Messages
4000 - 4999	Informational Messages

When the compiler encounters a severe error, it displays the message and stops processing. When the compiler encounters other types of errors, it displays the error message and continues processing. For any input file containing a severe error or an error, the compiler will not produce an output (.RES) file.

Severe Errors: 1000 - 1999

R1001 RC cannot open file *filename* .

Explanation: Resource Compiler was unable to open the given file.

Recovery: Check the format of the file. The same file might be in use by another process.

R1002 RC could not find file *filename* .

Explanation: Resource Compiler was unable to locate the given file.

Recovery: Check the spelling of the file name and its path, and be sure that the file exists.

R1003 RC had an I/O error with file *filename* .

Explanation: Resource Compiler was unable to read or write the given file.

Recovery: Check the spelling of the file name and its path. If the file exists, confirm that its format is correct.

R1004 You did not specify an input file.

Explanation: You did not name an input file on the command line.

Recovery: Correct the syntax of your command line.

R1005 You did not specify an output file.

Explanation: You did not name an output file on the command line, in a context which required an output file name.

Recovery: Correct the syntax of your command line.

R1006 The system DBCS environment is not correct.

Explanation: Resource Compiler detected an error when examining your system's double-byte character set environment vector.

Recovery: Correct your system's double-byte character set environment.

R1007 You specified more than one output file on the command line.

Explanation: You specified extra file operands following the output file name on your command line.

Recovery: Correct the syntax of your command line.

R1008 The input and output file names must be different; both are *filename* .

Explanation: On the command line, your input and output file names were identical.

Recovery: Give a different name for the output file.

R1009 RC detected errors during compilation.

Explanation: The compiler was unable to complete the compile process.

Recovery: Correct the errors noted on your display.

R1010 RC read an unexpected end of file on *filename* .

Explanation: File *filename* did not contain certain data that Resource Compiler needed.

Recovery: Correct the contents of the file.

R1012 RC has an internal data error.

Explanation: Resource Compiler detected an incorrect data state.

Recovery: Follow your local problem reporting procedures to notify IBM of the error.

R1013 RC encountered errors in input file *filename* ; resources will not be bound.

Explanation: Resource Compiler could not continue the compile process.

Recovery: Correct the file according to the error messages printed.

R1014 RC has an internal logic error -- consult your IBM representative.

Explanation: Resource Compiler detected an incorrect internal state.

Recovery: Follow your local problem reporting procedures to notify IBM of the error.

R1015 The executable file was marked not loadable.

Explanation: You requested that Resource Compiler bind resources to an executable file, but the file could not be loaded.

Recovery: Use a loadable executable file for binding resources.

R1016 RC failed to allocate memory.

Explanation: A request for system memory could not be satisfied.

Recovery: Your system's swapper file might not have enough disk space to grow to a necessary size. Clear some space on the device containing your swapper file.

R1017 Not enough memory is available.

Explanation: Tried an operation that use more than the available memory.

Recovery: Stop some programs to give more memory.

R1018 RC cannot open the file.

Explanation: Resource Compiler attempted a file I/O operation which was unsuccessful.

Recovery: Check the spelling of the file name and its path, and be sure that the file exists. Be sure that other processes have released the file for operations by the Resource Compiler.

R1019 RC cannot read from the file.

Explanation: Resource Compiler attempted a file I/O operation which was unsuccessful.

Recovery: Check the spelling of the file name and its path, and be sure that the file exists. Be sure that other processes have released the file for operations by the Resource Compiler.

R1020 RC cannot write to the file.

Explanation: Resource Compiler attempted a file I/O operation which was unsuccessful.

Recovery: Check the spelling of the file name and its path, and be sure that the file exists. Be sure that other processes have released the file for operations by the Resource Compiler.

R1021 RC cannot seek to a file.

Explanation: Resource Compiler attempted a file I/O operation which was unsuccessful.

Recovery: Check the spelling of the file name and its path, and be sure that the file exists. Be sure that other processes have released the file for operations by the Resource Compiler.

R1022 The file has no data.

Explanation: Resource Compiler attempted to read an empty file.

Recovery: Confirm that your input files have the proper data format.

R1025 Codepage could not be found on this system.

Explanation: The code page specified does not exist.

Recovery: Specify an existing code page.

Error Messages: 2000 - 2999

R2002 RC expected at least one argument for *option* .

Explanation: You coded command-line option *option* without argument.

Recovery: Supply the required argument for the option.

R2003 This input is incorrect:

Explanation: The given text is incorrect.

Recovery: Correct the syntax of the statement.

R2004 RC encountered an incorrect character in the source file.

Explanation: The input file had a character which was incorrect in context.

Recovery: Remove the incorrect character from the file.

R2009 You defined too many macros from the command line.

Explanation: You specified more than eight macro definitions on the command line.

Recovery: Supply extra macro definitions as #define statements within an input file.

R2010 You specified the incorrect code page value *value* .

Explanation: You coded command-line option -cp or -k or environment variable DBCS but gave an incorrect value for the code page.

Recovery: Use a value from the table of code pages and country codes in "COUNTRYCODE" in the *OS/2 Warp Control Program Programming Guide and Reference* .

R2011 You specified no code page value.

Explanation: You coded command-line option -cp or -k or environment variable DBCS but gave no value for the code page.

Recovery: Use a value from the table of code pages and country codes in "COUNTRYCODE" in the *OS/2 Warp Control Program Programming Guide and Reference* .

R2013 The resource identifiers have too many long strings.

Explanation: Your resource data contained more than 64K bytes of strings used as resource identifiers. The resources could not be bound to the executable file.

Recovery: Use fewer characters in the string identifiers of your resource file.

R2014 RC cannot delete old resources within instance pages.

Explanation: The input executable file contained original resources inside instance pages.

Recovery: Do not link resources into the executable file prior to running Resource Compiler.

R2015 The resources have too many objects.

Explanation: The input executable file contained an incorrect format.

Recovery: Correct the object table of the input file.

R2016 The page table has too many entries.

Explanation: The input executable file contained too many pages.

Recovery: Correct the page table of the input file.

R2017 The executable file has too many fixups.

Explanation: The input executable file contained too many fixups.

Recovery: Correct the fixup page table of the input file.

R2018 RC cannot write a resident Format Directive.

Explanation: The input executable file contained resident format directives which the compiler could not rewrite during the binding operation.

Recovery: Correct the format directives table of the input file.

R2019 RC cannot read the OS/2 segment table in file *filename* .

Explanation: The input executable file was not in the correct format.

Recovery: Correct the segment table of the input file.

R2020 RC cannot delete temporary file *filename* .

Explanation: The temporary executable file could not be deleted.

Recovery: Be sure that you have specified a value for the TMP or TEMP environment variable.

R2021 RC cannot write extended attributes for *filename* .

Explanation: Resource Compiler was unable to write extended attributes to the output executable file.

Recovery: Be sure that no other process is using the executable file.

R2022 The default icon exceeds 64K.

Explanation: The input executable file contained a default icon in excess of 64K in size.

Recovery: Use a default icon smaller than 64K.

R2023 The association table exceeds 64K.

Explanation: The input executable file contained an association table in excess of 64K in size.

Recovery: Use an association table smaller than 64K.

R2024 You must specify an executable file name.

Explanation: You did not specify the name of an executable file to which to bind resources.

Recovery: Specify the name of an executable file.

R2025 You must specify a resource or script file name.

Explanation: You did not specify the name of an input file.

Recovery: Specify the name of an input file.

R2026 RC encountered an error while binding resources to executable file *filename* .

Explanation: During the binding step, Resource Compiler found errors described in other messages.

Recovery: Follow the recovery steps given for the accompanying messages.

R2027 RC encountered an error while reading the resource file.

Explanation: Resource Compiler had an I/O error on the binary resource file.

Recovery: Confirm the spelling of the file name and path, and be sure that no other process is using a file of that name.

R2028 RC encountered an error while writing the resource file.

Explanation: Resource Compiler had an I/O error on the binary resource file.

Recovery: Confirm the spelling of the file name and path, and be sure that no other process is using a file of that name.

R2030 You must specified a resource and a script file name.

Explanation: Resource Compiler could not determine the names of the input script file and the output binary file.

Recovery: Give names for both of these files on the command line.

R2031 The executable file has an unknown format.

Explanation: You tried to bind resources to an executable file in an unknown format.

Recovery: Use an executable file in either the NE (16-bit) or LX (32-bit) linear executable format.

R2033 You specified the duplicate operand, which already exist in the group.

Explanation: You used a command line operand more than once.

Recovery: Remove the extra operand specification.

R2034 RC cannot create resource item type '*type* ' and id '*number* '.

Explanation: One of your resource statements specified an incorrect combination of resource type and resource ID.

Recovery: Do not duplicate resource IDs of the same type.

R2035 RC cannot create resource item type '*type* ' and id '*string* '.

Explanation: One of your resource statements specified an incorrect combination of resource type and resource ID.

Recovery: Do not duplicate resource IDs of the same type.

R2036 This resource name is too long (limit 512):

Explanation: You coded the given name as a resource id, but its length exceeds the legal limit.

Recovery: Use a resource name shorter than 512 characters.

R2037 RC failed to add a resource. The return code is *nnnnn* .

Explanation: An error occurred while creating a resource.

Recovery: Refer to the OS/2 return code *nnnnn* for more information.

R2038 The script file has an incorrect integer or hex or octal literal.

Explanation: You provided a numeric literal in an incorrect format, or with a value out of the legal range.

Recovery: Correct the format or value of the literal.

R2039 The script file has an incorrect floating point literal.

Explanation: You provided a numeric literal in an incorrect format, or with a value out of the legal range.

Recovery: Correct the format or value of the literal.

R2040 The script file has an incorrect floating point value.

Explanation: You provided a numeric literal in an incorrect format, or with a value out of the legal range.

Recovery: Correct the format or value of the literal.

R2041 The script file has an integer literal too large for its type.

Explanation: You provided a numeric literal in an incorrect format, or with a value out of the legal range.

Recovery: Correct the format or value of the literal.

R2042 You specified a character literal with more than one byte.

Explanation: You provided a character literal in an incorrect format, or with a value out of the legal range.

Recovery: Correct the format or value of the literal.

R2043 You specified an empty character literal.

Explanation: You provided a character literal in an incorrect format, or with a value out of the legal range.

Recovery: Correct the format or value of the literal.

R2044 The script file has a Hex or Octal escape sequence out of range.

Explanation: You provided an escape sequence in an incorrect format, or with a value out of the legal range.

Recovery: Correct the format or value of the sequence.

R2045 A Hex escape sequence contained no hex digits.

Explanation: You provided an escape sequence in an incorrect format, or with a value out of the legal range.

Recovery: Correct the format or value of the sequence.

R2046 You specified an incorrect escape sequence.

Explanation: You provided an escape sequence in an incorrect format, or with a value out of the legal range.

Recovery: Correct the format or value of the sequence.

R2047 The given directive requires a valid macro name.

Explanation: You specified preprocessor directive on which the expected macro name was missing or incorrect.

Recovery: Correct the macro name on the directive.

R2048 You specified an incorrect expression specified on a conditional directive.

Explanation: You specified an incorrect expression on an #if or #elif preprocessor directive.

Recovery: Correct the syntax of the expression.

R2049 The expression on a conditional directive results in division by zero.

Explanation: You specified an incorrect expression on an #if or #elif preprocessor directive.

Recovery: Correct the syntax of the expression.

R2050 #elif or #else or #endif appeared in an incorrect context.

Explanation: You specified an #elif or #else or #endif preprocessor directive without a preceding #if or #ifdef or #ifndef preprocessor directive.

Recovery: Correct the logic of the preprocessor conditional blocks.

R2051 The preprocessor expected #endif but the input stream ended with open scope.

Explanation: You specified #if or #elif or #ifdef or #ifndef preprocessor directive without a concluding #endif preprocessor directive.

Recovery: Correct the logic of the preprocessor conditional blocks.

R2052 You did not complete a #define parameter list before the end of the source line.

Explanation: You specified a function-type macro definition in a #define preprocessor directive but did not end the parameter list with a closing parenthesis.

Recovery: Correct the syntax of the macro definition.

R2053 You specified an incorrect #define parameter list.

Explanation: You specified a function-type macro definition in a #define preprocessor directive but did not end the parameter list with a closing parenthesis.

Recovery: Correct the syntax of the macro definition.

R2054 #define parameter list has a duplicate parameter.

Explanation: You repeated a parameter when specifying a function-type macro.

Recovery: Correct the syntax of the macro definition.

R2055 You invoked a macro with incorrect arguments.

Explanation: You specified incorrect arguments to a macro with a function-type definition.

Recovery: Correct the syntax of the macro reference.

R2056 You did not complete a directive before the end of the source line.

Explanation: You specified a preprocessor directive but did not complete it on the same line.

Recovery: Write the entire directive on the same line.

R2057 You did not end a character or string or #include literal before the end of the line.

Explanation: You specified a string or literal token which did not end before the line ended.

Recovery: Write the entire token on the same line.

R2058 An rcinclude statement must have a filename to include.

Explanation: You specified an RCINCLUDE statement but omitted the file name.

Recovery: Code the name of the file on the RCINCLUDE statement.

R2059 A statement has an incorrect expression.

Explanation: You specified an statement with an incorrect arithmetic expression.

Recovery: Correct the syntax of the expression.

R2060 The stringizing operator(#) appeared on a function-like macro.

Explanation: You specified a stringizing operator on a function-like macro.

Recovery: Refer to C Language Reference for the rules governing the stringizing operator.

R2061 You specified the Concatenation operator (##) incorrectly.

Explanation: You specified a concatenation operator in an incorrect context.

Recovery: Refer to C Language Reference for the rules governing the concatenation operator.

R2062 The script file has an incorrect wide character or wide string literal.

Explanation: You specified a wide character or wide string literal in an incorrect format.

Recovery: Refer to C Language Reference for the rules governing wide characters and wide strings.

R2063 A literal has an incorrect multibyte character.

Explanation: You specified a literal containing an incorrect multibyte character.

Recovery: Use only characters from your multibyte character set in multibyte literal constants.

R2064 You used a wide character string on an #include.

Explanation: You specified an #include preprocessor directive containing a wide character string. This usage is prohibited.

Recovery: Correct the syntax of the directive.

R2065 A predefined macro can not be defined again.

Explanation: You used a #define or #undef preprocessor directive to redefine a predefined macro. These macro names cannot be changed using directives.

Recovery: For control over the values of macros, use macro names which are not reserved for special use by the preprocessor.

R2066 A predefined macro can not be undefined.

Explanation: You used a #define or #undef preprocessor directive to undefine a predefined macro. These macro names cannot be changed using directives.

Recovery: For control over the values of macros, use macro names which are not reserved for special use by the preprocessor.

R2067 The filename to be searched contained only whitespace.

Explanation: You used an #include preprocessor directive to process a file which had no tokens.

Recovery: Eliminate the reference to the file, or supply the correct information inside it.

R2068 The file cannot be searched because of missing include options.

Explanation: You asked for a file to be included on an #include directive and wrote the file name without a path name inside angle brackets, and gave no path search specifications with the -I option. With no search criteria, the preprocessor will be unable to locate the file.

Recovery: Specify search paths to the file by using the -I option.

R2069 You redefined a macro name with different replacement text.

Explanation: You defined a macro, then redefined it to a different value.

Recovery: To change the value of a macro, first undefine it and then use another #define directive.

R2076 An unexpected newline appeared within quotes.

Explanation: You defined a string spanning more than one line, when this usage is not permitted.

Recovery: Remove the newline from the string.

R2077 A Hex escape must be followed by 2 or 4 hex digits.

Explanation: You used a hexadecimal escape sequence in the wrong format.

Recovery: Only one-byte and two-byte characters are recognized as hex escapes. For example, use the form \x00 for a single-byte zero or \x0000 for a double-byte zero.

R2078 A resource_Item has incorrect Window Data.

Explanation: Resource Compiler encountered incorrect binary data while examining the child windows of a resource.

Recovery: The message appears together with other messages describing the particular errors of the child window. Follow the recovery instructions of the accompanying messages.

R2079 You specified a non-numeric button item style.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2080 You specified a non-numeric combo box style.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2081 You specified a non-numeric control style.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2082 You specified a non-numeric entry field style.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2083 You specified a non-numeric frame style.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2084 You specified a non-numeric list box style.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2085 You specified a non-numeric static style.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2086 You specified a non-numeric window style.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2087 You specified a non-numeric control id.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2088 You specified a non-numeric resource id.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2089 You specified a non-numeric frame control flag.

Explanation: You coded a field using a string token.

Recovery: Replace the string by a defined numeric constant or expression.

R2090 You specified an incorrect window class.

Explanation: You coded a window class field using an unknown string or number.

Recovery: Use a correct value for the window class.

R2091 You specified an incorrect code page.

Explanation: You coded a code page field using an unknown number.

Recovery: Use a value from the table of code pages and country codes in "COUNTRYCODE" in the *OS/2 Warp Control Program Programming Guide and Reference* .

R2092 RC is using code page *codepage* .

Explanation: You coded a code page field that is not defined. The compiler chose *codepage* as the code page value.

Recovery: Use a value from the table of code pages and country codes in "COUNTRYCODE" in the *OS/2 Warp Control Program Programming Guide and Reference* .

R2093 You specified an incorrect memory or load option.

Explanation: You used an incorrect memory or load option.

Recovery: Valid memory options are FIXED, MOVEABLE, and DISCARDABLE. Valid load options are PRELOAD and LOADONCALL.

R2094 You specified a non-numeric menu item style.

Explanation: You used an incorrect menu item style.

Recovery: Use a style as documented under MENUITEM Statement.

R2096 The value of the height must be from 0 to 65535.

Explanation: You used an incorrect token in a numeric field.

Recovery: Confirm that each macro you used has a definition. You might have omitted a field from a list of parameters separated by commas.

R2097 You specified a non-numeric height.

Explanation: You used an incorrect token in a numeric field.

Recovery: Confirm that each macro you used has a definition. You might have omitted a field from a list of parameters separated by commas.

R2098 The value of width must be from 0 to 65535.

Explanation: You used an incorrect token in a numeric field.

Recovery: Confirm that each macro you used has a definition. You might have omitted a field from a list of parameters separated by commas.

R2099 You specified a non-numeric width.

Explanation: You used an incorrect token in a numeric field.

Recovery: Confirm that each macro you used has a definition. You might have omitted a field from a list of parameters separated by commas.

R2100 The value of X Coordinate must be from -32768 to 32767.

Explanation: You used an incorrect token in a numeric field.

Recovery: Confirm that each macro you used has a definition. You might have omitted a field from a list of parameters separated by commas.

R2101 You specified a non-numeric X Coordinate.

Explanation: You used an incorrect token in a numeric field.

Recovery: Confirm that each macro you used has a definition. You might have omitted a field from a list of parameters separated by commas.

R2102 The value of Y Coordinate must be from -32768 to 32767.

Explanation: You used an incorrect token in a numeric field.

Recovery: Confirm that each macro you used has a definition. You might have omitted a field from a list of parameters separated by commas.

R2103 You specified a non-numeric Y Coordinate.

Explanation: You used an incorrect token in a numeric field.

Recovery: Confirm that each macro you used has a definition. You might have omitted a field from a list of parameters separated by commas.

R2106 A left parenthesis is incorrect or missing.

Explanation: You used a numeric expression which lacked balanced parentheses.

Recovery: Correct the parentheses in the expression.

R2108 The value of a custom-resource type must be from 1 to 65535.

Explanation: A RESOURCE statement contained a type field which evaluated to a number outside the legal range.

Recovery: Use a value from 1 to 65535 as a resource type.

R2109 You specified an incorrect Assocname in ASSOCTABLE.

Explanation: An ASSOCTABLE statement contained an incorrect field as indicated.

Recovery: Use a legal value for the field. The syntax rules appear under ASSOCTABLE Statement.

R2110 You specified an incorrect File-match-string in ASSOCTABLE.

Explanation: An ASSOCTABLE statement contained an incorrect field as indicated.

Recovery: Use a legal value for the field. The syntax rules appear under ASSOCTABLE Statement.

R2111 You specified an incorrect ea-flag in ASSOCTABLE.

Explanation: An ASSOCTABLE statement contained an incorrect field as indicated.

Recovery: Use a legal value for the field. The syntax rules appear under ASSOCTABLE Statement.

R2112 An iconfile parameter is missing from the ASSOCTABLE.

Explanation: An ASSOCTABLE statement contained an incorrect field as indicated.

Recovery: Use a legal value for the field. The syntax rules appear under ASSOCTABLE Statement.

R2113 You specified an incorrect parameter in SUBITEMSIZE.

Explanation: A SUBITEMSIZE statement contained an incorrect value as indicated.

Recovery: Use a legal value for the field. The syntax rules appear under SUBITEMSIZE Statement.

R2114 The size value in SUBITEMSIZE must be 2 or greater.

Explanation: A SUBITEMSIZE statement contained an incorrect value as indicated.

Recovery: Use a legal value for the field. The syntax rules appear under SUBITEMSIZE Statement.

R2115 A font file has an incorrect kern pair identifier or non-zero usKerningPairs.

Explanation: A FONT or RESOURCE statement referred to a font file which has an incorrect binary format.

Recovery: Correct the kerning information table in the referenced font file.

R2116 A font file has an incorrect additional metrics identifier or font signature.

Explanation: A FONT statement referred to a file containing incorrect data as indicated.

Recovery: Use a properly-formatted font file.

R2117 You specified an incorrect data type in RCDATA.

Explanation: An RCDATA statement contained an incorrect numeric or string data item.

Recovery: Correct the syntax of the item. The syntax rules appear under RCDATA Statement.

R2118 You specified an incorrect data type in RESOURCE.

Explanation: RESOURCE statement contained an incorrect numeric or string data item.

Recovery: Correct the syntax of the item. The syntax rules appear under RESOURCE Statement.

R2119 You specified an incorrect text string or ordinal.

Explanation: A statement contained an incorrect numeric or string data item.

Recovery: Correct the syntax of the item.

R2120 The value of an ordinal must be from 0 to 65535.

Explanation: A statement contained an incorrect numeric or string data item.

Recovery: Correct the syntax of the item.

R2121 A logical OR cannot follow another logical OR.

Explanation: A statement contained an incorrect expression with logical operators.

Recovery: Correct the syntax of the expression.

R2122 An integer value must follow the not operator.

Explanation: A statement contained an incorrect expression with logical operators.

Recovery: Correct the syntax of the expression.

R2123 RC found an incorrect character in an Expression.

Explanation: A statement contained an expression with incorrect characters.

Recovery: Correct the syntax of the expression.

R2124 An ordinal value must be preceded by a #.

Explanation: A MENUITEM statement with style MIS_BITMAP accepts in its text field a quoted value of a previously-defined bitmap, in the form "#n", where n is the bitmap id. You coded this field in an incorrect format as indicated in the message.

Recovery: Correct the syntax of the text field.

R2125 You must specify an ordinal value from 0 to 65535 after a '#'.

Explanation: A MENUITEM statement with style MIS_BITMAP accepts in its text field a quoted value of a previously-defined bitmap, in the form "#n", where n is the bitmap id. You coded this field in an incorrect format as indicated in the message.

Recovery: Correct the syntax of the text field.

R2126 You specified an incorrect text parameter.

Explanation: A statement contained a text field with an incorrect string.

Recovery: Correct the syntax of the field.

R2127 You specified a non-numeric parameter.

Explanation: A statement contained a numeric field that did not evaluate to a number.

Recovery: Correct the syntax of the field.

R2128 The Resource ID must be a value from 1 to 65535 (unsigned).

Explanation: A statement contained a resource ID field that did not evaluate to a number in the expected range.

Recovery: Correct the value of the id field.

R2129 The Resource ID must be a value from 0 to 65535 (unsigned) or -32768 to 32767 (signed).

Explanation: A statement contained a resource ID field that did not evaluate to a number in the expected range.

Recovery: Correct the value of the id field.

R2130 The Resource ID must not be negative.

Explanation: A statement contained a resource ID field that did not evaluate to a number in the expected range.

Recovery: Correct the value of the id field.

R2131 The Resource ID must not be 0.

Explanation: A statement contained a resource ID field that did not evaluate to a number in the expected range.

Recovery: Correct the value of the id field.

R2132 The Control ID must be a value from 0 to 65535 (unsigned) or -32768 to 32767 (signed).

Explanation: A statement contained a resource ID field that did not evaluate to a number in the expected range.

Recovery: Correct the value of the id field.

R2134 Only one top level window is allowed.

Explanation: A WINDOWTEMPLATE or DLGTEMPLATE statement contained more than one WINDOW or DIALOG.

Recovery: Use only one window per template.

R2135 A Template statement is empty. You must specify a top level window.

Explanation: A WINDOWTEMPLATE or DLGTEMPLATE statement contained no WINDOW or DIALOG.

Recovery: Use exactly one window per template.

R2136 RC found a duplicate string id.

Explanation: A STRINGTABLE statement contained a duplicate identifier.

Recovery: Use unique string ids.

R2137 RC found a duplicate message id.

Explanation: A MESSAGETABLE statement contained a duplicate identifier.

Recovery: Use unique and message ids.

R2138 A key-value character code in double quotation marks is a missing parameter.

Explanation: An ACCELTABLE statement contained a key-value field in an incorrect format as indicated by the message.

Recovery: Correct the format or value of the field. The syntax rules appear under ACCELTABLE Statement.

R2139 Parameter key-value has more than the allowed characters.

Explanation: An ACCELTABLE statement contained a key-value field in an incorrect format as indicated by the message.

Recovery: Correct the format or value of the field. The syntax rules appear under ACCELTABLE Statement.

R2140 Parameter key-value with control (^) is out of the valid range (^A - ^Z).

Explanation: An ACCELTABLE statement contained a key-value field in an incorrect format as indicated by the message.

Recovery: Correct the format or value of the field. The syntax rules appear under ACCELTABLE Statement.

R2141 Parameter key-value must be in the range (0 - 255).

Explanation: An ACCELTABLE statement contained a key-value field in an incorrect format as indicated by the message.

Recovery: Correct the format or value of the field. The syntax rules appear under ACCELTABLE Statement.

R2142 Parameter key-value must be string or numeric.

Explanation: An ACCELTABLE statement contained a key-value field in an incorrect format as indicated by the message.

Recovery: Correct the format or value of the field. The syntax rules appear under ACCELTABLE Statement.

R2143 Parameter command must be numeric.

Explanation: An ACCELTABLE statement contained a command field in an incorrect format as indicated by the message.

Recovery: Correct the format or value of the field. The syntax rules appear under ACCELTABLE Statement.

R2144 Parameter command must be a value from 0 to 65535.

Explanation: An ACCELTABLE statement contained a command field in an incorrect format as indicated by the message.

Recovery: Correct the format or value of the field. The syntax rules appear under ACCELTABLE Statement.

R2145 Parameter accel-option is incorrect.

Explanation: An ACCELTABLE statement contained a accel-option field in an incorrect format (or missing) as indicated by the message.

Recovery: Supply the correct value of the field. The syntax rules appear under ACCELTABLE Statement.

R2146 Accel-option (CHAR, SCANCODE, or VIRTUALKEY) is a missing parameter.

Explanation: An ACCELTABLE statement contained a accel-option field in an incorrect format (or missing) as indicated by the message.

Recovery: Supply the correct value of the field. The syntax rules appear under ACCELTABLE Statement.

R2147 The accelerator table is empty.

Explanation: An ACCELTABLE statement contained an empty data block.

Recovery: Supply the missing table. The syntax rules appear under ACCELTABLE Statement.

R2149 You specified a non-numeric menu item attribute.

Explanation: A MENUITEM statement contained an undefined attribute.

Recovery: Supply the missing attribute value. The syntax rules appear under MENUITEM Statement.

R2150 The menu is empty.

Explanation: A MENU statement contained an empty data block.

Recovery: Supply the missing menu items. The syntax rules appear under MENU Statement.

R2151 You specified an incorrect presentation parameter.

Explanation: A control statement contained an incorrect presentation parameter.

Recovery: Correct syntax of the value or parameter.

R2152 You specified an incorrect control data value.

Explanation: A control statement contained an incorrect control data value.

Recovery: Correct syntax of the value or parameter.

R2153 DLGINCLUDE statement must have a filename to include.

Explanation: You specified a DLGINCLUDE statement but omitted the file name.

Recovery: Code the name of the file on the DLGINCLUDE statement.

R2154 Data in a STRINGTABLE statement BEGIN/END body is missing or incorrect.

Explanation: You specified incorrect string data inside the data block of a STRINGTABLE statement, as indicated by the message.

Recovery: Correct the syntax of the string. The syntax rules appear under STRINGTABLE Statement.

R2155 Data in a MESSAGETABLE statement BEGIN/END body is missing or incorrect.

Explanation: You specified incorrect string data inside the data block of a MESSAGETABLE statement, as indicated by the message.

Recovery: Correct the syntax of the string. The syntax rules appear under MESSAGETABLE Statement.

R2156 An Assocname is missing in ASSOCTABLE.

Explanation: The field indicated by the message is missing from an ASSOCTABLE statement.

Recovery: Supply the missing field.

R2157 A file-match-string is missing in ASSOCTABLE.

Explanation: The field indicated by the message is missing from an ASSOCTABLE statement.

Recovery: Supply the missing field.

R2158 Data in an RCDATA statement BEGIN/END body is missing or incorrect.

Explanation: Data items are wrong or omitted in the statement indicated by the message.

Recovery: Supply the correct data.

R2159 Data in RESOURCE statement BEGIN/END body is missing or incorrect.

Explanation: Data items are wrong or omitted in the statement indicated by the message.

Recovery: Supply the correct data.

R2160 Data in ASSOCTABLE statement BEGIN/END body is missing or incorrect.

Explanation: Data items are wrong or omitted in the statement indicated by the message.

Recovery: Supply the correct data.

R2161 RC expected a SUBITEMSIZE statement before the parameter.

Explanation: A HELPSUBTABLE statement was missing the SUBITEMSIZE statement required when a size other than the default is to be used.

Recovery: Supply the missing SUBITEMSIZE statement. The syntax rules appear under HELPSUBTABLE Statement

R2162 Elements in HELPSUBITEM do not match with SUBITEMSIZE statement.

Explanation: The structure of a HELPSUBITEM statement did not conform to the size specified in the SUBITEMSIZE statement.

Recovery: Correct the format of the HELPSUBITEM statement. The syntax rules appear under HELPSUBITEM Statement

2163 The Resource Type value overflowed when creating the LR formatted record.

Explanation: A Resource Type cannot exceed 65535.

Recovery: Specify a value between 0 and 65535.

2164 The Resource ID value overflowed when creating the LR formatted record.

Explanation: A Resource ID cannot exceed 65535.

Recovery: Specify a value between 0 and 65535.

2165 The Resource Flag value overflowed when creating the LR formatted record.

Explanation: A Resource Flag cannot exceed 65535.

Recovery: Specify a value between 0 and 65535.

2166 The Resource Size value overflowed when creating the LR formatted record.

Explanation: A Resource Size cannot exceed 65535.

Recovery: Specify a value between 0 and 65535.

R2167 Data in MSGDATA statement BEGIN/END body is missing or incorrect.

Explanation: The MSGDATA body is missing or incorrect.

Recovery: Correct the syntax or add the missing data.

R2168 A suffix letter is missing after the message id.

Explanation: You left out a required suffix letter after the message id.

Recovery: Correct your input and try again.

R2169 A colon (:) is missing after the suffix letter.

Explanation: You left out a required colon (:) after the suffix letter.

Recovery: Correct your input and try again.

R2170 A MSGDATA statement has a duplicate message ID.

Explanation: You cannot use the same message ID on more than one MSGDATA statements.

Recovery: Use a unique message ID.

R2171 The segment name contains extended characters, which must be less than or equal to ASCII 127.

Explanation: Segment names must use ASCII characters with a value less than or equal to 127.

Recovery:

R2172 A Resource statement must have either a file specification or a BEGIN/END block.

Explanation: You coded a RESOURCE statement without a file specification or a data block, or you provided both a file specification and a data block. Exactly one of these forms must be provided.

Recovery: Correct the format of the RESOURCE statement. The syntax rules appear under

R2173 The source file ended with a continuation sequence.

Explanation: Your input script file ended prematurely before the end of a continued line.

Recovery: Supply the completion of the incomplete line.

R2174 The source file ended while a comment was active.

Explanation: Your input script file ended prematurely before the end of a comment.

Recovery: Supply the completion of the incomplete comment.

R2175 You specified an unknown preprocessor directive.

Explanation: You used a # directive which is unknown to the preprocessor.

Recovery: Remove the unknown directive.

R2176 A header filename was missing from an #include directive.

Explanation: You used an #include directive without naming a file to include.

Recovery: Supply the missing file name on the directive.

R2177 A header filename contained only whitespace.

Explanation: The file referenced by an #include directive contained no tokens to parse.

Recovery: Supply the missing contents of the file, or remove the directive.

R2178 RC could not find the header filename on an #include directive.

Explanation: The file referenced by an #include directive could not be located by the preprocessor.

Recovery: Check the spelling of the file and path, and confirm that the file exists.

R2179 #undef appeared with unknown macro name.

Explanation: The macro referenced by an #undef directive was not known to the preprocessor.

Recovery: Check the spelling of the macro, or remove the #undef directive.

R2180 You specified an include filename recursively.

Explanation: The name on an #include directive has already been included at a higher level.

Recovery: Remove the nested #include directive.

R2181 A source filename contains no characters.

Explanation: The input file was empty.

Recovery: Check the spelling of the name, and confirm that the proper data are contained in it.

R2182 RC is ignoring a typecast in an expression.

Explanation: You used a type cast in an expression in a preprocessor directive. The cast will be ignored.

Recovery: Remove the type cast.

R2183 RC found the characters /* inside a C style comment.

Explanation: The input file had a C comment which contained the start of another comment.

Recovery: Do not use the characters /* inside a C comment; comments cannot be nested.

R2184 You defined a macro with the name of a resource statement.

Explanation: You used the name of a resource statement (for example, search file for more e.g.'s. DLGTEMPLATE) as the name of a macro to be defined. The names of resource statements are reserved words.

Recovery: Use another name for the macro.

R2185 The given directive is missing its end-of-line character.

Explanation: You used a preprocessor directive (such as #define) but started another statement later on the same line. A preprocessor directive needs to be contained on one line.

Recovery: Use the preprocessor directive alone on a line.

R2186 You specified conflicting or redundant memory or load options.

Explanation: Memory or load options you specified conflict or are redundant.

Recovery: Specify non-conflicting or non-redundant memory or load options.

Warning Messages: 3000 - 3999

R3006 *name* is an incorrect macro name.

Explanation: The macro name *name* does not conform to the rules for composing macro names as defined in C Language Reference. The Resource Compiler will ignore the definition.

Recovery: Choose a legal name for the macro.

R3007 Country code *code* is incorrect for this code page. RC is using previously-defined lead bytes.

Explanation: The country *code* is not defined for use with the specified code page.

Recovery: Confirm that the country code and code page are correct. For a list of supported code pages, refer to the "COUNTRYCODE" section of the *OS/2 Warp Control Programming Guide and Reference* .

R3008 Code page *cp* is incorrect. RC is using the default code page.

Explanation: The code page number *cp* is unknown, and the compiler will use a default code page.

Recovery: Select a supported code page. For a list of supported code pages, refer to the "COUNTRYCODE" section of the *OS/2 Warp Control Program Programming Guide and Reference* .

R3009 RC is ignoring excess operand *text* .

Explanation: The given *text* was an incorrect operand on the command line.

Recovery: Correct or remove the undefined text.

R3011 RC is ignoring the compression option for the NE format executable file *filename* .

Explanation: The compression option you specified is not valid for this operation.

Recovery: No action is necessary.

R3012 RC is ignoring the pack option for the NE executable file *filename* .

Explanation: The pack option you specified is not a valid for this operation.

Recovery: No action is necessary.

R3013 RC found no resource item in this file.

Explanation: The input binary resource file did not contain any resources.

Recovery: Confirm that the input file has the expected resource data.

R3014 RC found a resource type equal to zero.

Explanation: The input binary resource file contained a resource with type zero, which is incorrect.

Recovery: Correct the resource definition in the input file.

R3015 RC is using ordinal *number* as a text field.

Explanation: The specified *number* is the value of a text field on a resource statement.

Recovery: No action is necessary.

R3016 RC is truncating this text to 255 characters:

Explanation: The text that follows this message appeared in a field limited to 255 characters. The compiler will use only the text shown.

Recovery: No action is necessary.

R3017 RC is ignoring an extra comma in a CTLDATA statement.

Explanation: You used a CTLDATA statement with a list of data values separated by commas. The compiler will ignore the extra commas it found in the list.

Recovery: No action is necessary.

R3018 RC is truncating to a 16-bit value.

Explanation: You used a long integer value in a data field which is limited to 64K. The compiler will use only the least significant 16 bits of the number.

Recovery: No action is necessary.

R3019 All text after percent followed by 0 characters will be truncated to the end of the line.

Explanation: You enter a text string with a percent 0 (%0) in it.

Recovery: No action is necessary.

R3020 Parameters VIRTUALKEY, SCANMODE, and CHAR are mutually exclusive.

Explanation: You used parameters in an ACCELTABLE statement which cannot be used together.

Recovery: The rules for coding the accelerator-options field are given under

R3021 Parameters SYSCOMMAND, and HELP are mutually exclusive.

Explanation: You used parameters in an ACCELTABLE statement which cannot be used together.

Recovery: The rules for coding the accelerator-options field are given under ACCELTABLE Statement.

R3022 Numeric value in RCDATA statement has been overflowed.

Explanation: While evaluating numeric data in a block, the compiler computed a 32-bit number using a 16-bit data field. This conversion changed the value of the number.

Recovery: No action is necessary.

R3023 Numeric value in RCDATA statement has been modified.

Explanation: While evaluating numeric data in a block, the compiler computed a 32-bit number using a 16-bit data field. This conversion changed the value of the number.

Recovery: No action is necessary.

R3024 Numeric value in RESOURCE statement has been overflowed.

Explanation: While evaluating numeric data in a block, the compiler computed a 32-bit number using a 16-bit data field. This conversion changed the value of the number.

Recovery: No action is necessary.

R3025 Numeric value in RESOURCE statement has been modified.

Explanation: While evaluating numeric data in a block, the compiler computed a 32-bit number using a 16-bit data field. This conversion changed the value of the number.

Recovery: No action is necessary.

R3026 MENUITEM expected a style of MIS_SUBMENU.

Explanation: A MENUITEM statement having a submenu will use menuitem style MIS_SUBMENU by default.

Recovery: Specify MIS_SUBMENU in MENUITEM statement.

R3027 RC ignored text specified for SEPARATOR.

Explanation: A MENUITEM SEPARATOR statement had extra text at the end. Resource Compiler will ignore the extra text.

Recovery: Remove the extra text.

R3028 RC is ignoring a Message text line with no message ID.

Explanation: A MESSAGETABLE statement without an ID has been ignored by the Resource Compiler.

Recovery: To use the message, provide an identifier for it.

R3030 RC encountered a duplicate resource type and id. RC is ignoring resource ID *id* of type *type* .

Explanation: A resource of type *type* and id *id* has the same type and id as another resource. The Resource Compiler used the first and ignored the other.

Recovery: Provide a unique identifier for the duplicated resource.

R3031 The ID value to be used is *value* .

Explanation: The preprocessor will use *value* as the resource identifier.

Recovery: No action is necessary.

R3032 A parameter value is out of range. Valid signed range is -32768 to 32767, unsigned range is 0 to 65535.

Explanation: You supplied a value for a 16-bit parameter which was too large to be contained in the field.

Recovery: Resource Compiler has converted the value to the 16-bit number given.

R3033 RC is using the first DEFAULTICON statement, which you specified more than once.

Explanation: You supplied more than one DEFAULTICON statement for your resources. Resource Compiler will ignore all DEFAULTICON statements after the first one.

Recovery: Remove the extra DEFAULTICON statements.

R3034 *'text'* is not supported in this version of RC. It is being ignored.

Explanation: The given text is not supported by this version of the Resource Compiler. It is ignored.

Recovery: Remove the text.

R3035 The DBCS lead-byte option is no longer supported in RC.

Explanation: The command line and DBCS environment variable options will be ignored.

Recovery: Use an installed code page.

Informational Messages: 4000 - 4999

R4012 Writing resources to OS/2 v2.0 Linear .EXE file

Explanation: The Resource Compiler is binding resources to an executable file.

Recovery: No action is necessary.

R4013 RC is copying *numpages* pages: *numobjects* objects containing *numpages* page(s).

Explanation: The program is performing the action you requested.

Recovery: No action is necessary.

R4014 RC is writing *nnnn* resource object(s).

Explanation: The program is performing the action you requested.

Recovery: No action is necessary.

R4015 RC is writing *numbytes* bytes in *numpages* page(s).

Explanation: The program is performing the action you requested.

Recovery: No action is necessary.

R4019 Writing Extended Attributes:

Explanation: The Resource Compiler program is writing the extended attributes.

Recovery: No action is necessary.

R4020 Writing Extended Attributes: Checksum.

Explanation: The Resource Compiler program is writing the extended attributes: Checksum.

Recovery: No action is necessary.

R4023 RC successfully created a preprocessed output file.

Explanation: The preprocessor wrote its output file to disk.

Recovery: No action is necessary.

Notices

April 2001

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

This publication was developed for products and services offered in the United States of America. IBM may not offer the products, services, or features discussed in this document in other countries, and the information is subject to change without notice. Consult your local IBM representative for information on the products, services, and features available in your area.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Copyright Notices

Disclaimers

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department LZKS, 11400 Burnet Road, Austin, TX 78758 U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX
C Set ++
Common User Access
CUA
IBM
Operating System/2
OS/2
Personal System/2
PowerPC
Presentation Manager
SAA
System Application Architecture
WIN-OS/2
Workplace Shell
XGA

The following terms are trademarks of other companies:

CL, CL386
Intel
Intel Corporation
MASM, MASM386
Pentium
X/Open Company Ltd.
/X/Opend

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

C-bus is a trademark of Corollary, Inc.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

This glossary defines many of the terms used in this book. It includes terms and definitions from the *IBM Dictionary of Computing* , as well as terms specific to the OS/2 operating system and the Presentation Manager. It is not a complete glossary for the entire OS/2 operating system; nor is it a complete dictionary of computer terms.

Other primary sources for these definitions are:

- The *American National Standard Dictionary for Information Systems* , ANSI X3.172-1990, copyrighted 1990 by the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. These definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary* , developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international

standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Glossary Listing

Select a starting letter of glossary terms:

Glossary - A

accelerator - In SAA Common User Access architecture, a key or combination of keys that invokes an application-defined function.

accelerator table - A table used to define which key strokes are treated as *accelerators* and the commands they are translated into.

access mode - The manner in which an application gains access to a file it has opened. Examples of access modes are read-only, write-only, and read/write.

access permission - All access rights that a user has regarding an object. (I)

action - One of a set of defined tasks that a computer performs. Users request the application to perform an action in several ways, such as typing a command, pressing a function key, or selecting the action name from an action bar or menu.

action bar - In SAA Common User Access architecture, the area at the top of a window that contains choices that give a user access to actions available in that window.

action point - The current position on the screen at which the pointer is pointing. Contrast with *hot spot* and *input focus* .

active program - A program currently running on the computer. An active program can be interactive (running and receiving input from the user) or noninteractive (running but not receiving input from the user). See also *interactive program* and *noninteractive program* .

active window - The window with which the user is currently interacting.

address space - (1) The range of addresses available to a program. (A) (2) The area of virtual storage available for a particular job.

alphanumeric video output - Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

American National Standard Code for Information Interchange - The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

Note: IBM has defined an extension to ASCII code (characters 128-255).

anchor - A window procedure that handles Presentation Manager message conversions between an icon procedure and an application.

anchor block - An area of Presentation-Manager-internal resources to allocated process or thread that calls WinInitialize.

anchor point - A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI - American National Standards Institute.

APA - All points addressable.

API - Application programming interface.

application - A collection of software components used to perform specific types of work on a computer; for example, a payroll application, an airline reservation application, a network application.

application object - In SAA Advanced Common User Access architecture, a form that an application provides for a user; for example, a spreadsheet form. Contrast with *user object* .

application programming interface (API) - A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

application-modal - Pertaining to a message box or dialog box for which processing must be completed before further interaction with any other window owned by the same application may take place.

area - In computer graphics, a filled shape such as a solid rectangle.

ASCII - American National Standard Code for Information Interchange.

ASCIIZ - A string of ASCII characters that is terminated with a byte containing the value 0.

aspect ratio - In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous (ASYNC) - (1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (T) (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions. See also *synchronous* .

atom - A constant that represents a string. As soon as a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an *atom table* . See also *integer atom* .

atom table - A table used to relate *atoms* with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

atomic operation - An operation that completes its work on an object before another operation can be performed on the same object.

attribute - A characteristic or property that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes* .

automatic link - In Information Presentation Facility (IPF), a link that begins a chain reaction at the primary window. When the user selects the primary window, an automatic link is activated to display secondary windows.

AVIO - Advanced Video Input/Output.

Glossary - B

Bézier curve - (1) A mathematical technique of specifying smooth continuous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bézier. (2) (D of C) In the AIX Graphics Library, a cubic spline approximation to a set of four control points that passes through the first and fourth control points and that has a continuous slope where two spline segments meet. Named after Dr. P. Bézier.

background - (1) In multiprogramming, the conditions under which low-priority programs are executed. Contrast with *foreground* . (2) An active session that is not currently displayed on the screen.

background color - The color in which the background of a graphic primitive is drawn.

background mix - An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with *mix* .

background program - In multiprogramming, a program that executes with a low priority. Contrast with *foreground program* .

bit map - A representation in memory of the data displayed on an APA device, usually the screen.

block - (1) A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or logical records. (T) (2) To record data in a block. (3) A collection of contiguous records recorded as a unit. Blocks are separated by interblock gaps and each block may contain one or more records. (A)

block device - A storage device that performs I/O operations on blocks of data called *sectors* . Data on block devices can be randomly accessed. Block devices are designated by a drive letter (for example, **C:**).

blocking mode - A condition set by an application that determines when its threads might block. For example, an application might set the Pipemode parameter for the DosCreateNPipe function so that its threads perform I/O operations to the named pipe block when no data is available.

border - A visual indication (for example, a separator line or a background color) of the boundaries of a window.

boundary determination - An operation used to compute the size of the smallest rectangle that encloses a graphics object on the screen.

breakpoint - (1) A point in a computer program where execution may be halted. A breakpoint is usually at the beginning of an instruction where halts, caused by external intervention, are convenient for resuming execution. (T) (2) A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

broken pipe - When all of the handles that access one end of a pipe have been closed.

bucket - One or more fields in which the result of an operation is kept.

buffer - (1) A portion of storage used to hold input or output data temporarily. (2) To allocate and schedule the use of buffers. (A)

button - A mechanism used to request or initiate an action. See also *barrel buttons* , *bezel buttons* , *mouse button* , *push button* , and *radio button* .

byte pipe - Pipes that handle data as byte streams. All unnamed pipes are byte pipes. Named pipes can be byte pipes or message pipes. See *byte stream* .

byte stream - Data that consists of an unbroken stream of bytes.

Glossary - C

cache - A high-speed buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.

cached micro presentation space - A presentation space from a Presentation-Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

CAD - Computer-Aided Design.

call - (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (I) (A) (2) To transfer control to a procedure, program, routine, or subroutine.

calling sequence - A sequence of instructions together with any associated data necessary to execute a call. (T)

Cancel - An action that removes the current window or menu without processing it, and returns the previous window.

cascaded menu - In the OS/2 operating system, a menu that appears when the arrow to the right of a cascading choice is selected. It contains a set of choices that are related to the cascading choice. Cascaded menus are used to reduce the length of a menu. See also *cascading choice* .

cascading choice - In SAA Common User Access architecture, a choice in a menu that, when selected, produces a cascaded menu containing other choices. An arrow ( ) appears to the right of the cascading choice.

CASE statement - In PM programming, provides the body of a window procedure. There is usually one CASE statement for each message type supported by an application.

CGA - Color graphics adapter.

chained list - A list in which the data elements may be dispersed but in which each data element contains information for locating the next. (T) Synonymous with *linked list* .

character - A letter, digit, or other symbol.

character box - In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also *character mode* . Contrast with *character cell* .

character cell - The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box* .

character code - The means of addressing a character in a character set, sometimes called *code point* .

character device - A device that performs I/O operations on one character at a time. Because character devices view data as a stream of bytes, character-device data cannot be randomly accessed. Character devices include the keyboard, mouse, and printer, and are referred to by name.

character mode - A mode that, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

character set - (1) An ordered set of unique representations called characters; for example, the 26 letters of English alphabet, Boolean 0 and 1, the set of symbols in the Morse code, and the 128 ASCII characters. (A) (2) All the valid characters for a programming language or for a computer system. (3) A group of characters used for a specific reason; for example, the set of characters a printer can print.

check box - In SAA Advanced Common User Access architecture, a square box with associated text that represents a choice. When a user selects a choice, an X appears in the check box to indicate that the choice is in effect. The user can clear the check box by selecting the choice again. Contrast with *radio button* .

check mark - (1) (D of C) In SAA Advanced Common User Access architecture, a symbol that shows that a choice is currently in effect. (2) The symbol that is used to indicate a selected item on a pull-down menu.

child process - In the OS/2 operating system, a process started by another process, which is called the parent process. Contrast with *parent process* .

child window - A window that appears within the border of its parent window (either a primary window or another child window). When the parent window is resized, moved, or destroyed, the child window also is resized, moved, or destroyed; however, the child window can be moved or resized independently from the parent window, within the boundaries of the parent window. Contrast with *parent window* .

choice - (1) An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol). (2) (D of C) In SAA Common User Access architecture, an item that a user can select.

chord - (1) To press more than one button on a pointing device while the pointer is within the limits that the user has specified for the operating environment. (2) (D of C) In graphics, a short line segment whose end points lie on a circle. Chords are a means for producing a circular image from straight lines. The higher the number of chords per circle, the smoother the circular image.

class - A way of categorizing objects based on their behavior and shape. A class is, in effect, a definition of a generic object. In SOM, a class is a special kind of object that can manufacture other objects that all have a common shape and exhibit similar behavior (more precisely, all of the objects manufactured by a class have the same memory layout and share a common set of methods). New classes can be defined in terms of existing classes through a technique known as *inheritance* .

class method - A class method of class <X> is a method provided by the metaclass of class <X>. Class methods are executed without requiring any instances of class <X> to exist, and are frequently used to create instances. In System Object Model, an action that can be performed on a class object.

class object - In System Object Model, the run-time implementation of a class.

class style - The set of properties that apply to every window in a window class.

client - (1) A functional unit that receives shared services from a server. (T) (2) A user, as in a client process that uses a named pipe or queue that is created and owned by a server process.

client area - The part of the window, inside the border, that is below the menu bar. It is the user's work space, where a user types information and selects choices from selection fields. In primary windows, it is where an application programmer presents the objects that a user works on.

client program - An application that creates and manipulates instances of classes.

client window - The window in which the application displays output and receives input. This window is located inside the frame window, under the window title bar and any menu bar, and within any scroll bars.

clip limits - The area of the paper that can be reached by a printer or plotter.

clipboard - In SAA Common User Access architecture, an area of computer memory, or storage, that temporarily holds data. Data in the clipboard is available to other applications.

clipping - In computer graphics, removing those parts of a display image that lie outside a given boundary. (I) (A)

clipping area - The area in which the window can paint.

clipping path - A clipping boundary in world-coordinate space.

clock tick - The minimum unit of time that the system tracks. If the system timer currently counts at a rate of X Hz, the system tracks the time every 1/X of a second. Also known as *time tick* .

CLOCK$ - Character-device name reserved for the system clock.

code page - An assignment of graphic characters and control-function meanings to all code points.

code point - (1) Synonym for *character code* . (2) (D of C) A 1-byte code representing one of 256 potential characters.

code segment - An executable section of programming code within a load module.

color dithering - See *dithering* .

color graphics adapter (CGA) - An adapter that simultaneously provides four colors and is supported by all IBM Personal Computer and Personal System/2 models.

command - The name and parameters associated with an action that a program can perform.

command area - An area composed of a command field prompt and a command entry field.

command entry field - An entry field in which users type commands.

command line - On a display screen, a display line, sometimes at the bottom of the screen, in which only commands can be entered.

command mode - A state of a system or device in which the user can enter commands.

command prompt - A field prompt showing the location of the command entry field in a panel.

Common Programming Interface (CPI) - Definitions of those application development languages and services that have, or are intended to have, implementations on and a high degree of commonality across the SAA environments. One of the three SAA architectural areas. See also *Common User Access architecture* .

Common User Access (CUA) architecture - Guidelines for the dialog between a human and a workstation or terminal. One of the three SAA architectural areas. See also *Common Programming Interface* .

compile - To translate a program written in a higher-level programming language into a machine language program.

composite window - A window composed of other windows (such as a frame window, frame-control windows, and a client window) that are kept together as a unit and that interact with each other.

computer-aided design (CAD) - The use of a computer to design or change a product, tool, or machine, such as using a computer for drafting or illustrating.

COM1, COM2, COM3 - Character-device names reserved for serial ports 1 through 3.

CON - Character-device name reserved for the console keyboard and screen.

conditional cascaded menu - A pull-down menu associated with a menu item that has a cascade mini-push button beside it in an object's pop-up menu. The conditional cascaded menu is displayed when the user selects the mini-push button.

container - In SAA Common User Access architecture, an object that holds other objects. A folder is an example of a container object. See also *folder* and *object* .

contextual help - In SAA Common User Access Architecture, help that gives specific information about the item the cursor is on. The help is contextual because it provides information about a specific item as it is currently being used. Contrast with *extended help* .

contiguous - Touching or joining at a common edge or boundary, for example, an unbroken consecutive series of storage locations.

control - In SAA Advanced Common User Access architecture, a component of the user interface that allows a user to select choices or type information; for example, a check box, an entry field, a radio button.

control area - A storage area used by a computer program to hold control information. (I) (A)

Control Panel - In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program - (1) The basic functions of the operating system, including DOS emulation and the support for keyboard, mouse, and video input/output. (2) A computer program designed to schedule and to supervise the execution of programs of a computer system. (I) (A)

control window - A window that is used as part of a composite window to perform simple input and output tasks. Radio buttons and check boxes are examples.

control word - An instruction within a document that identifies its parts or indicates how to format the document.

coordinate space - A two-dimensional set of points used to generate output on a video display of printer.

Copy - A choice that places onto the clipboard, a copy of what the user has selected. See also *Cut* and *Paste* .

correlation - The action of determining which element or object within a picture is at a given position on the display. This follows a *pick* operation.

coverpage window - A window in which the application's help information is displayed.

CPI - Common Programming Interface.

critical extended attribute - An extended attribute that is necessary for the correct operation of the system or a particular application.

critical section - (1) In programming languages, a part of an asynchronous procedure that cannot be executed simultaneously with a certain part of another asynchronous procedure. (I)

Note: Part of the other asynchronous procedure also is a critical section. (2) A section of code that is not reentrant; that is, code that can be executed by only one thread at a time.

CUA architecture - Common User Access architecture.

current position - In computer graphics, the position, in user coordinates, that becomes the starting point for the next graphics routine, if that routine does not explicitly specify a starting point.

cursor - A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with *pointer* and *input focus* .

Cut - In SAA Common User Access architecture, a choice that removes a selected object, or a part of an object, to the clipboard, usually compressing the space it occupied in a window. See also *Copy* and *Paste* .

Glossary - D

daisy chain - A method of device interconnection for determining interrupt priority by connecting the interrupt sources serially.

data segment - A nonexecutable section of a program module; that is, a section of a program that contains data definitions.

data structure - The syntactic structure of symbolic expressions and their storage-allocation characteristics. (T)

data transfer - The movement of data from one object to another by way of the clipboard or by direct manipulation.

DBCS - Double-byte character set.

DDE - Dynamic data exchange.

deadlock - (1) Unresolved contention for the use of a resource. (2) An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by, or a response from, the other. (3) An impasse that occurs when multiple processes are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar wait state.

debug - To detect, diagnose, and eliminate errors in programs. (T)

decipoint - In printing, one tenth of a point. There are 72 points in an inch.

default procedure - A function provided by the Presentation Manager Interface that may be used to process standard messages from

dialogs or windows.

default value - A value assumed when no value has been specified. Synonymous with assumed value. For example, in the graphics programming interface, the default line-type is 'solid'.

definition list - A type of list that pairs a term and its description.

delta - An application-defined threshold, or number of container items, from either end of the list.

descendant - See *child process* .

descriptive text - Text used in addition to a field prompt to give more information about a field.

Deselect all - A choice that cancels the selection of all of the objects that have been selected in that window.

Desktop Manager - In the Presentation Manager, a window that displays a list of groups of programs, each of which can be started or stopped.

desktop window - The window, corresponding to the physical device, against which all other types of windows are established.

detached process - A background process that runs independent of the parent process.

detent - A point on a slider that represents an exact value to which a user can move the slider arm.

device context - A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also *direct device context* , *information device context* , *memory device context* , *metafile device context* , *queued device context* , and *screen device context* .

device driver - A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space - (1) Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units. (2) (D of C) In computer graphics, a space defined by the complete set of addressable points of a display device. (A)

dialog - The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box - In SAA Advanced Common User Access architecture, a movable window, fixed in size, containing controls that a user uses to provide information required by an application so that it can continue to process a user request. See also *message box, primary window, secondary window* . Also known as a *pop-up window* .

Dialog Box Editor - A *WYSIWYG* editor that creates dialog boxes for communicating with the application user.

dialog item - A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog procedure - A dialog window that is controlled by a window procedure. It is responsible for responding to all messages sent to the dialog window.

dialog tag language - A markup language used by the DTL compiler to create dialog objects.

dialog template - The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context - A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also *device context* .

direct manipulation - The user's ability to interact with an object by using the mouse, typically by dragging an object around on the Desktop and dropping it on other objects.

direct memory access (DMA) - A technique for moving data directly between main storage and peripheral equipment without requiring processing of the data by the processing unit.(T)

directory - A type of file containing the names and controlling information for other files or other directories.

display point - Synonym for *pel* .

dithering - (1) The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work, for example, on narrow lines. (2) (D of C) In computer graphics, a technique of interleaving dark and light pixels so that the resulting image looks smoothly shaded when viewed from a distance.

DMA - Direct memory access.

DOS Protect Mode Interface (DPMI) - An interface between protect mode and real mode programs.

double-byte character set (DBCS) - A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

doubleword - A contiguous sequence of bits or characters that comprises two computer words and is capable of being addressed as a unit. (A)

DPMI - DOS Protect Mode Interface.

drag - In SAA Common User Access, to use a pointing device to move an object; for example, clicking on a window border, and dragging it to make the window larger.

dragging - (1) In computer graphics, moving an object on the display screen as if it were attached to the pointer. (2) (D of C) In computer graphics, moving one or more segments on a display surface by translating. (I) (A)

drawing chain - See *segment chain* .

drop - To fix the position of an object that is being dragged, by releasing the select button of the pointing device. See also *drag* .

DTL - Dialog tag language.

dual-boot function - A feature of the OS/2 operating system that allows the user to start DOS from within the operating system, or an OS/2 session from within DOS.

duplex - Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full duplex* .

dynamic data exchange (DDE) - A message protocol used to communicate between applications that share data. The protocol uses shared memory as the means of exchanging data between applications.

dynamic data formatting - A formatting procedure that enables you to incorporate text, bit maps or metafiles in an IPF window at execution time.

dynamic link library - A collection of executable programming code and data that is bound to an application at load time or run time, rather than during linking. The programming code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic linking - The process of resolving external references in a program module at load time or run time rather than during linking.

dynamic segments - Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

dynamic storage - (1) A device that stores data in a manner that permits the data to move or vary with time such that the specified data is not always available for recovery. (A) (2) A storage in which the cells require repetitive application of control signals in order to retain stored data. Such repetitive application of the control signals is called a refresh operation. A dynamic storage may use static addressing or sensing circuits. (A) (3) See also *static storage* .

dynamic time slicing - Varies the size of the time slice depending on system load and paging activity.

dynamic-link module - A module that is linked at load time or run time.

Glossary - E

EBCDIC - Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters (9 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

edge-triggered - Pertaining to an event semaphore that is posted then reset before a waiting thread gets a chance to run. The semaphore is considered to be posted for the rest of that thread's waiting period; the thread does not have to wait for the semaphore to be posted again.

EGA - Extended graphics adapter.

element - An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

EMS - Expanded Memory Specification.

encapsulation - Hiding an object's implementation, that is, its private, internal data and methods. Private variables and methods are accessible only to the object that contains them.

entry field - In SAA Common User Access architecture, an area where a user types information. Its boundaries are usually indicated. See also *selection field* .

entry panel - A defined panel type containing one or more entry fields and protected information such as headings, prompts, and explanatory text.

entry-field control - The component of a user interface that provides the means by which the application receives data entered by the user in an entry field. When it has the input focus, the entry field displays a flashing pointer at the position where the next typed character will go.

environment segment - The list of environment variables and their values for a process.

environment strings - ASCII text strings that define the value of environment variables.

environment variables - Variables that describe the execution environment of a process. These variables are named by the operating system or by the application. Environment variables named by the operating system are PATH, DPATH, INCLUDE, INIT, LIB, PROMPT, and TEMP. The values of environment variables are defined by the user in the CONFIG.SYS file, or by using the SET command at the OS/2 command prompt.

error message - An indication that an error has been detected. (A)

event semaphore - A semaphore that enables a thread to signal a waiting thread or threads that an event has occurred or that a task has been completed. The waiting threads can then perform an action that is dependent on the completion of the signaled event.

exception - An abnormal condition such as an I/O error encountered in processing a data set or a file.

exclusive system semaphore - A system semaphore that can be modified only by threads within the same process.

executable file - (1) A file that contains programs or commands that perform operations or actions to be taken. (2) A collection of related data records that execute programs.

exit - To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T) Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel* .

expanded memory specification (EMS) - Enables DOS applications to access memory above the 1MB real mode addressing limit.

extended attribute - An additional piece of information about a file object, such as its data format or category. It consists of a name and a value. A file object may have more than one extended attribute associated with it.

extended help - In SAA Common User Access architecture, a help action that provides information about the contents of the application window from which a user requested help. Contrast with *contextual help* .

extended-choice selection - A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection* .

extent - Continuous space on a disk or diskette that is occupied by or reserved for a particular data set, data space, or file.

external link - In Information Presentation Facility, a link that connects external online document files.

Glossary - F

family-mode application - An application program that can run in the OS/2 environment and in the DOS environment; however, it cannot take advantage of many of the OS/2-mode facilities, such as multitasking, interprocess communication, and dynamic linking.

FAT - File allocation table.

FEA - Full extended attribute.

field-level help - Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used; the information is dependent upon the context within the work session.

FIFO - First-in-first-out. (A)

file - A named set of records stored or processed as a unit. (T)

file allocation table (FAT) - In IBM personal computers, a table used by the operating system to allocate space on a disk for a file, and to locate and chain together parts of the file that may be scattered on different sectors so that the file can be used in a random or sequential manner.

file attribute - Any of the attributes that describe the characteristics of a file.

File Manager - In the Presentation Manager, a program that displays directories and files, and allows various actions on them.

file specification - The full identifier for a file, which includes its drive designation, path, file name, and extension.

file system - The combination of software and hardware that supports storing information on a storage device.

file system driver (FSD) - A program that manages file I\O and controls the format of information on the storage media.

fillet - A curve that is tangential to the end points of two adjoining lines. See also *polyfillet* .

filtering - An application process that changes the order of data in a queue.

first-in-first-out (FIFO) - A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

flag - (1) An indicator or parameter that shows the setting of a switch. (2) A character that signals the occurrence of some condition, such as the end of a word. (A) (3) (D of C) A characteristic of a file or directory that enables it to be used in certain ways. See also *archive* flag, *hidden flag* , and *read-only flag* .

focus - See *input focus* .

folder - A container used to organize objects.

font - A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

Font Editor - A utility program provided with the IBM Developers Toolkit that enables the design and creation of new fonts.

foreground program - (1) The program with which the user is currently interacting. Also known as *interactive program* . Contrast with *background program* . (2) (D of C) In multiprogramming, a high-priority program.

frame - The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles - Standard window layouts provided by the Presentation Manager.

FSD - File system driver.

full-duplex - Synonym for *duplex* .

full-screen application - An application that has complete control of the screen.

function - (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a call. (2) A set of related control statements that cause one or more programs to be performed.

function key - A key that causes a specified sequence of operations to be performed when it is pressed, for example, F1 and Alt-K.

function key area - The area at the bottom of a window that contains function key assignments such as F1=Help.

Glossary - G

GDT - Global Descriptor Table.

general protection fault - An exception condition that occurs when a process attempts to use storage or a module that has some level of protection assigned to it, such as I/O privilege level. See also *IOPL code segment* .

Global Descriptor Table (GDT) - A table that defines code and data segments available to all tasks in an application.

global dynamic-link module - A dynamic-link module that can be shared by all processes in the system that refer to the module name.

global file-name character - Either a question mark (?) or an asterisk (*) used as a variable in a file name or file name extension when referring to a particular file or group of files.

glyph - A graphic symbol whose appearance conveys information.

GPI - Graphics programming interface.

graphic primitive - In computer graphics, a basic element, such as an arc or a line, that is not made up of smaller parts and that is used to create diagrams and pictures. See also *graphics segment* .

graphics - (1) A picture defined in terms of graphic primitives and graphics attributes. (2) (D of C) The making of charts and pictures. (3) Pertaining to charts, tables, and their creation. (4) See *computer graphics, coordinate graphics, fixed-image graphics, interactive graphics, passive graphics, raster graphics* .

graphics attributes - Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also *segment attributes* .

graphics field - The clipping boundary that defines the visible part of the presentation-page contents.

graphics mode - One of several states of a display. The mode determines the resolution and color content of the screen.

graphics model space - The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as *model space* .

Graphics programming interface - The formally defined programming language that is between an IBM graphics program and the user of the program.

graphics segment - A sequence of related graphic primitives and graphics attributes. See also *graphic primitive* .

graying - The indication that a choice on a pull-down is unavailable.

group - A collection of logically connected controls. For example, the buttons controlling paper size for a printer could be called a group. See also *program group* .

Glossary - H

handle - (1) An identifier that represents an object, such as a device or window, to the Presentation Interface. (2) (D of C) In the Advanced DOS and OS/2 operating systems, a binary value created by the system that identifies a drive, directory, and file so that the file can be found and opened.

hard error - An error condition on a network that requires either that the system be reconfigured or that the source of the error be removed before the system can resume reliable operation.

header - (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message, such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

heading tags - A document element that enables information to be displayed in windows, and that controls entries in the contents window controls placement of push buttons in a window, and defines the shape and size of windows.

heap - An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

help function - (1) A function that provides information about a specific field, an application panel, or information about the help facility. (2) (D of C) One or more display images that describe how to use application software or how to do a system operation.

Help index - In SAA Common User Access architecture, a help action that provides an index of the help information available for an application.

help panel - A panel with information to assist users that is displayed in response to a help request from the user.

help window - A Common-User-Access-defined secondary window that displays information when the user requests help.

hidden file - An operating system file that is not displayed by a directory listing.

hide button - In the OS/2 operating system, a small, square button located in the right-hand corner of the title bar of a window that, when selected, removes from the screen all the windows associated with that window. Contrast with *maximize button* . See also *restore button* .

hierarchical inheritance - The relationship between parent and child classes. An object that is lower in the inheritance hierarchy than another object, inherits all the characteristics and behaviors of the objects above it in the hierarchy.

hierarchy - A tree of segments beginning with the root segment and proceeding downward to dependent segment types.

high-performance file system (HPFS) - In the OS/2 operating system, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. The file system also supports the coexistence of multiple, active file systems on a single personal computer, with the capability of multiple and different storage devices. File names used with the HPFS can have as many as 254 characters.

hit testing - The means of identifying which window is associated with which input device event.

hook - A point in a system-defined function where an application can supply additional code that the system processes as though it were part of the function.

hook chain - A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot spot - The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point* .

HPFS - high-performance file system.

hypergraphic link - A connection between one piece of information and another through the use of graphics.

hypertext - A way of presenting information online with connections between one piece of information and another, called *hypertext links* . See also *hypertext link* .

hypertext link - A connection between one piece of information and another.

Glossary - I

I/O operation - An input operation to, or output operation from a device attached to a computer.

I-beam pointer - A pointer that indicates an area, such as an entry field in which text can be edited.

icon - In SAA Advanced Common User Access architecture, a graphical representation of an object, consisting of an image, image background, and a label. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions, and minimized programs.

icon area - In the Presentation Manager, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor - The Presentation Manager-provided tool for creating icons.

IDL - Interface Definition Language.

image font - A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of one of the symbols. Contrast with *outline font* .

implied metaclass - Subclassing the metaclass of a parent class without a separate CSC for the resultant metaclass.

indirect manipulation - Interaction with an object through choices and controls.

information device context - A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also *device context* .

information panel - A defined panel type characterized by a body containing only protected information.

Information Presentation Facility (IPF) - A facility provided by the OS/2 operating system, by which application developers can produce online documentation and context-sensitive online help panels for their applications.

inheritance - The technique of specifying the shape and behavior of one class (called a *subclass*) as incremental differences from another class (called the *parent class* or *superclass*). The subclass inherits the superclass' state representation and methods, and can provide additional data elements and methods. The subclass also can provide new functions with the same method names used by the superclass. Such a subclass method is said to override the superclass method, and will be selected automatically by method resolution on subclass instances. An overriding method can elect to call upon the superclass' method as part of its own implementation.

input focus - (1) The area of a window where user interaction is possible using an input device, such as a mouse or the keyboard. (2) The position in the *active window* where a user's normal interaction with the keyboard will appear.

input router - An internal OS/2 process that removes messages from the system queue.

input/output control - A device-specific command that requests a function of a device driver.

installable file system (IFS) - A file system in which software is installed when the operating system is started.

instance - (Or object instance). A specific object, as distinguished from the abstract definition of an object referred to as its class.

instance method - A method valid for a particular object.

instruction pointer - In System/38, a pointer that provides addressability for a machine interface instruction in a program.

integer atom - An *atom* that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by Presentation Manager are expressed as integer atoms.

interactive graphics - Graphics that can be moved or manipulated by a user at a terminal.

interactive program - (1) A program that is running (active) and is ready to receive (or is receiving) input from a user. (2) A running program that can receive input from the keyboard or another input device. Compare with *active program* and contrast with *noninteractive program* .

Also known as a *foreground program* .

interchange file - A file containing data that can be sent from one Presentation Manager interface application to another.

Interface Definition Language (IDL) - Language-neutral interface specification for a SOM class.

interpreter - A program that translates and executes each instruction of a high-level programming language before it translates and executes.

interprocess communication (IPC) - In the OS/2 operating system, the exchange of information between processes or threads through semaphores, pipes, queues, and shared memory.

interval timer - (1) A timer that provides program interruptions on a program-controlled basis. (2) An electronic counter that counts intervals of time under program control.

IOCtl - Input/output control.

IOPL - Input/output privilege level.

IOPL code segment - An IOPL executable section of programming code that enables an application to directly manipulate hardware interrupts and ports without replacing the device driver. See also *privilege level* .

IPC - Interprocess communication.

IPF - Information Presentation Facility.

IPF compiler - A text compiler that interpret tags in a source file and converts the information into the specified format.

IPF tag language - A markup language that provides the instructions for displaying online information.

item - A data object that can be passed in a DDE transaction.

Glossary - J

journal - A special-purpose file that is used to record changes made in the system.

Glossary - K

Kanji - A graphic character set used in Japanese ideographic alphabets.

KBD$ - Character-device name reserved for the keyboard.

kernel - The part of an operating system that performs basic functions, such as allocating hardware resources.

kerning - The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

keyboard accelerator - A keystroke that generates a command message for an application.

keyboard augmentation - A function that enables a user to press a keyboard key while pressing a mouse button.

keyboard focus - A temporary attribute of a window. The window that has a keyboard focus receives all keyboard input until the focus changes to a different window.

Keys help - In SAA Common User Access architecture, a help action that provides a listing of the application keys and their assigned functions.

Glossary - L

label - In a graphics segment, an identifier of one or more elements that is used when editing the segment.

LAN - Local area network.

language support procedure - A function provided by the Presentation Manager Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

lazy drag - See *pickup and drop* .

lazy drag set - See *pickup set* .

LDT - In the OS/2 operating system, Local Descriptor Table.

LIFO stack - A stack from which data is retrieved in last-in, first-out order.

linear address - A unique value that identifies the memory object.

linked list - Synonym for *chained list* .

list box - In SAA Advanced Common User Access architecture, a control that contains scrollable choices from which a user can select one choice.

Note: In CUA architecture, this is a programmer term. The end user term is selection list.

list button - A button labeled with an underlined down-arrow that presents a list of valid objects or choices that can be selected for that field.

list panel - A defined panel type that displays a list of items from which users can select one or more choices and then specify one or more actions to work on those choices.

load time - The point in time at which a program module is loaded into main storage for execution.

load-on-call - A function of a linkage editor that allows selected segments of the module to be disk resident while other segments are executing. Disk resident segments are loaded for execution and given control when any entry point that they contain is called.

local area network (LAN) - (1) A computer network located on a user's premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. (T)

Note: A LAN does not use store and forward techniques. (2) A network in which a set of devices are connected to one another for communication and that can be connected to a larger network.

Local Descriptor Table (LDT) - Defines code and data segments specific to a single task.

lock - A serialization mechanism by means of which a resource is restricted for use by the holder of the lock.

logical storage device - A device that the user can map to a physical (actual) device.

LPT1, LPT2, LPT3 - Character-device names reserved for parallel printers 1 through 3.

Glossary - M

main window - The window that is positioned relative to the *desktop window* .

manipulation button - The button on a pointing device a user presses to directly manipulate an object.

map - (1) A set of values having a defined correspondence with the quantities or values of another set. (I) (A) (2) To establish a set of values having a defined correspondence with the quantities or values of another set. (I)

marker box - In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol - A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

marquee box - The rectangle that appears during a selection technique in which a user selects objects by drawing a box around them with a pointing device.

Master Help Index - In the OS/2 operating system, an alphabetic list of help topics related to using the operating system.

maximize - To enlarge a window to its largest possible size.

media window - The part of the physical device (display, printer, or plotter) on which a picture is presented.

memory block - Part memory within a heap.

memory device context - A logical description of a data destination that is a memory bit map. See also *device context* .

memory management - A feature of the operating system for allocating, sharing, and freeing main storage.

memory object - Logical unit of memory requested by an application, which forms the granular unit of memory manipulation from the application viewpoint.

menu - In SAA Advanced Common User Access architecture, an extension of the menu bar that displays a list of choices available for a selected choice in the menu bar. After a user selects a choice in menu bar, the corresponding menu appears. Additional pop-up windows can appear from menu choices.

menu bar - In SAA Advanced Common User Access architecture, the area near the top of a window, below the title bar and above the rest of the window, that contains choices that provide access to other menus.

menu button - The button on a pointing device that a user presses to view a pop-up menu associated with an object.

message - (1) In the Presentation Manager, a packet of data used for communication between the Presentation Manager interface and Presentation Manager applications (2) In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process.

message box - (1) A dialog window predefined by the system and used as a simple interface for applications, without the necessity of creating dialog-template resources or dialog procedures. (2) (D of C) In SAA Advanced Common User Access architecture, a type of window that shows messages to users. See also *dialog box, primary window, secondary window* .

message filter - The means of selecting which messages from a specific window will be handled by the application.

message queue - A sequenced collection of messages to be read by the application.

message stream mode - A method of operation in which data is treated as a stream of messages. Contrast with *byte stream* .

metacharacter - See *global file-name character* .

metaclass - A class whose instances are all classes. In SOM, any class descended from SOMClass is a metaclass. The methods of a metaclass are sometimes called "class" methods.

metafile - A file containing a series of attributes that set color, shape and size, usually of a picture or a drawing. Using a program that can interpret these attributes, a user can view the assembled image.

metafile device context - A logical description of a data destination that is a metafile, which is used for graphics interchange. See also *device context* .

metalanguage - A language used to specify another language. For example, data types can be described using a metalanguage so as to make the descriptions independent of any one computer language.

method - One of the units that makes up the behavior of an object. A method is a combination of a function and a name, such that many different functions can have the same name. Which function the name refers to at any point in time depends on the object that is to execute the method and is the subject of method resolution.

method override - The replacement, by a child class, of the implementation of a method inherited from a parent and an ancestor class.

mickey - A unit of measurement for physical mouse motion whose value depends on the mouse device driver currently loaded.

micro presentation space - A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize - To remove from the screen all windows associated with an application and replace them with an icon that represents the application.

mix - An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as *foreground mix* . Contrast with *background mix* .

mixed character string - A string containing a mixture of one-byte and *Kanji* or Hangeul (two-byte) characters.

mnemonic - (1) A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item. (2) (D of C) In SAA Advanced Common User Access architecture, usually a single character, within the text of a choice, identified by an underscore beneath the character. If all characters in a choice already serve as mnemonics for other choices, another character, placed in parentheses immediately following the choice, can be used. When a user types the mnemonic for a choice, the choice is either selected or the cursor is moved to that choice.

modal dialog box - In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that requires a user to enter information before continuing to work in the application window from which it was displayed. Contrast with *modeless dialog box* . Also known as a *serial dialog box* . Contrast with *parallel dialog box* .

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

model space - See *graphics model space* .

modeless dialog box - In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that allows users to continue their dialog with the application without entering information in the dialog box. Also known as a *parallel dialog box* . Contrast with *modal dialog box* .

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

module definition file - A file that describes the code segments within a load module. For example, it indicates whether a code segment is loadable before module execution begins (preload), or loadable only when referred to at run time (load-on-call).

mouse - In SAA usage, a device that a user moves on a flat surface to position a pointer on the screen. It allows a user to select a choice o function to be performed or to perform operations on the screen, such as dragging or drawing lines from one position to another.

MOUSE$ - Character-device name reserved for a mouse.

multiple-choice selection - In SAA Basic Common User Access architecture, a type of field from which a user can select one or more choices or select none. See also *check box* . Contrast with *extended-choice selection* .

multiple-line entry field - In SAA Advanced Common User Access architecture, a control into which a user types more than one line of information. See also *single-line entry field* .

multitasking - The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

mutex semaphore - (Mutual exclusion semaphore). A semaphore that enables threads to serialize their access to resources. Only the thread that currently owns the mutex semaphore can gain access to the resource, thus preventing one thread from interrupting operations being performed by another.

muxwait semaphore - (Multiple wait semaphore). A semaphore that enables a thread to wait either for multiple event semaphores to be posted or for multiple mutex semaphores to be released. Alternatively, a muxwait semaphore can be set to enable a thread to wait for any ONE of the event or mutex semaphores in the muxwait semaphore's list to be posted or released.

Glossary - N

named pipe - A named buffer that provides client-to-server, server-to-client, or full duplex communication between unrelated processes. Contrast with *unnamed pipe* .

national language support (NLS) - The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature, MRI, or documentation of a product.

nested list - A list that is contained within another list.

NLS - national language support.

non-8.3 file-name format - A file-naming convention in which file names can consist of up to 255 characters. See also *8.3 file-name* format.

noncritical extended attribute - An extended attribute that is not necessary for the function of an application.

nondestructive read - Reading that does not erase the data in the source location. (T)

noninteractive program - A running program that cannot receive input from the keyboard or other input device. Compare with *active program* , and contrast with *interactive program* .

nonretained graphics - Graphic primitives that are not remembered by the Presentation Manager interface when they have been drawn. Contrast with *retained graphics* .

null character (NUL) - (1) Character-device name reserved for a nonexistent (dummy) device. (2) (D of C) A control character that is used to accomplish media-fill or time-fill and that may be inserted into or removed from a sequence of characters without affecting the meaning of the sequence; however, the control of equipment or the format may be affected by this character. (I) (A)

null-terminated string - A string of (n+1) characters where the (n+1)th character is the 'null' character (0x00) Also known as 'zero-terminated' string and 'ASCIIZ' string.

Glossary - O

object - The elements of data and function that programs create, manipulate, pass as arguments, and so forth. An object is a way of associating specific data values with a specific set of named functions (called *methods*) for a period of time (referred to as the *lifetime* of the object). The data values of an object are referred to as its *state* . In SOM, objects are created by other objects called *classes* . The specification of what comprises the set of functions and data elements that make up an object is referred to as the *definition* of a class.

SOM objects offer a high degree of *encapsulation* . This property permits many aspects of the implementation of an object to change without affecting client programs that depend on the object's behavior.

object definition - See *class* .

object instance - See *instance* .

Object Interface Definition Language (OIDL) - Specification language used in SOM Version 1 for defining classes. Replaced by Interface Definition Language (IDL).

object window - A window that does not have a parent but which might have child windows. An object window cannot be presented on a device.

OIDL - Object Interface Definition Language.

open - To start working with a file, directory, or other object.

ordered list - Vertical arrangements of items, with each item in the list preceded by a number or letter.

outline font - A set of symbols, each of which is created as a series of lines and curves. Synonymous with *vector font* . Contrast with *image font* .

output area - An area of storage reserved for output. (A)

owner window - A window into which specific events that occur in another (owned) window are reported.

ownership - The determination of how windows communicate using messages.

owning process - The process that owns the resources that might be shared with other processes.

Glossary - P

page - (1) A 4KB segment of contiguous physical memory. (2) (D of C) A defined unit of space on a storage medium.

page viewport - A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint - (1) The action of drawing or redrawing the contents of a window. (2) In computer graphics, to shade an area of a display image; for example, with crosshatching or color.

panel - In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

panel area - An area within a panel that contains related information. The three major Common User Access-defined panel areas are the action bar, the function key area, and the panel body.

panel area separator - In SAA Basic Common User Access architecture, a solid, dashed, or blank line that provides a visual distinction between two adjacent areas of a panel.

panel body - The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body can contain protected information, selection fields, and entry fields. The layout and content of the panel body determine the panel type.

panel body area - See *client area* .

panel definition - A description of the contents and characteristics of a panel. A panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID - In SAA Basic Common User Access architecture, a panel identifier, located in the upper-left corner of a panel. A user can choose whether to display the panel ID.

panel title - In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

paper size - The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box - See *modeless dialog box* .

parameter list - A list of values that provides a means of associating addressability of data defined in a called program with data in the calling program. It contains parameter names and the order in which they are to be associated in the calling and called program.

parent class - See *inheritance* .

parent process - In the OS/2 operating system, a process that creates other processes. Contrast with *child process* .

parent window - In the OS/2 operating system, a window that creates a child window. The child window is drawn within the parent window. If the parent window is moved, resized, or destroyed, the child window also will be moved, resized, or destroyed. However, the child window can be moved and resized independently from the parent window, within the boundaries of the parent window. Contrast with *child window* .

partition - (1) A fixed-size division of storage. (2) On an IBM personal computer fixed disk, one of four possible storage areas of variable size; one may be accessed by DOS, and each of the others may be assigned to another operating system.

Paste - A choice in the Edit pull-down that a user selects to move the contents of the clipboard into a preselected location. See also *Copy* and *Cut* .

path - The route used to locate files; the storage location of a file. A fully qualified path lists the drive identifier, directory name, subdirectory name (if any), and file name with the associated extension.

PDD - Physical device driver.

peeking - An action taken by any thread in the process that owns the queue to examine queue elements without removing them.

pel - (1) The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for *display point* , *pixel* , and *picture element* . (2) (D of C) Picture element.

persistent object - An object whose instance data and state are preserved between system shutdown and system startup.

physical device driver (PDD) - A system interface that handles hardware interrupts and supports a set of input and output functions.

pick - To select part of a displayed object using the pointer.

pickup - To add an object or set of objects to the pickup set.

pickup and drop - A drag operation that does not require the direct manipulation button to be pressed for the duration of the drag.

pickup set - The set of objects that have been picked up as part of a pickup and drop operation.

picture chain - See *segment chain* .

picture element - (1) Synonym for *pel* . (2) (D of C) In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity. (T) . (3) The area of the finest detail that can be reproduced effectively on the recording medium.

PID - Process identification.

pipe - (1) A named or unnamed buffer used to pass data between processes. A process reads from or writes to a pipe as if the pipe were a standard-input or standard-output file. See also *named pipe* and *unnamed pipe* . (2) (D of C) To direct data so that the output from one process becomes the input to another process. The standard output of one command can be connected to the standard input of another with the pipe operator (|).

pixel - (1) Synonym for *pel* . (2) (D of C) Picture element.

plotter - An output unit that directly produces a hardcopy record of data on a removable medium, in the form of a two-dimensional graphic representation. (T)

PM - Presentation Manager.

pointer - (1) The symbol displayed on the screen that is moved by a pointing device, such as a *mouse* . The pointer is used to point at items that users can select. Contrast with *cursor* . (2) A data element that indicates the location of another data element. (T)

POINTER$ - Character-device name reserved for a pointer device (mouse screen support).

pointing device - In SAA Advanced Common User Access architecture, an instrument, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

pointings - Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a *mouse* .

polyfillet - A curve based on a sequence of lines. The curve is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also *fillet* .

polygon - One or more closed figures that can be drawn filled, outlined, or filled and outlined.

polyline - A sequence of adjoining lines.

polymorphism - The ability to have different implementations of the same method for two or more classes of objects.

pop - To retrieve an item from a last-in-first-out stack of items. Contrast with *push* .

pop-up menu - A menu that lists the actions that a user can perform on an object. The contents of the pop-up menu can vary depending on the context, or state, of the object.

pop-up window - (1) A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window. (2) (D of C) In SAA Advanced Common User Access architecture, a movable window, fixed in size, in which a user provides information required by an application so that it can continue to process a user request.

presentation drivers - Special purpose I/O routines that handle field device-independent I/O requests from the PM and its applications.

Presentation Manager (PM) - The interface of the OS/2 operating system that presents, in windows a graphics-based interface to applications and files installed and running under the OS/2 operating system.

presentation page - The coordinate space in which a picture is assembled for display.

presentation space (PS) - (1) Contains the device-independent definition of a picture. (2) (D of C) The display space on a display device.

primary window - In SAA Common User Access architecture, the window in which the main interaction between the user and the application takes place. In a multiprogramming environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward. See also *secondary window* .

primitive - In computer graphics, one of several simple functions for drawing on the screen, including, for example, the rectangle, line, ellipse, polygon, and so on.

primitive attribute - A specifiable characteristic of a graphic primitive. See *graphics attributes* .

print job - The result of sending a document or picture to be printed.

Print Manager - In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

privilege level - A protection level imposed by the hardware architecture of the IBM personal computer. There are four privilege levels (number 0 through 3). Only certain types of programs are allowed to execute at each privilege level. See also */OPL code segment* .

procedure call - In programming languages, a language construct for invoking execution of a procedure.

process - An instance of an executing application and the resources it is using.

program - A sequence of instructions that a computer can interpret and execute.

program details - Information about a program that is specified in the *Program Manager* window and is used when the program is started.

program group - In the Presentation Manager, several programs that can be acted upon as a single entity.

program name - The full file specification of a program. Contrast with *program title* .

program title - The name of a program as it is listed in the *Program Manager* window. Contrast with *program name* .

prompt - A displayed symbol or message that requests input from the user or gives operational information; for example, on the display screen of an IBM personal computer, the DOS A> prompt. The user must respond to the prompt in order to proceed.

protect mode - A method of program operation that limits or prevents access to certain instructions or areas of storage. Contrast with *real mode* .

protocol - A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. (I)

pseudocode - An artificial language used to describe computer program algorithms without using the syntax of any particular programming language. (A)

pull-down - (1) An *action bar* extension that displays a list of choices available for a selected action bar choice. After users select an action bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the actions available to users. (2) (D of C) In SAA Common User Access architecture, pertaining to a choice in an action bar pull-down.

push - To add an item to a last-in-first-out stack of items. Contrast with *pop* .

push button - In SAA Advanced Common User Access architecture, a rectangle with text inside. Push buttons are used in windows for actions that occur immediately when the push button is selected.

putback - To remove an object or set of objects from the lazy drag set. This has the effect of undoing the pickup operation for those objects

putdown - To drop the objects in the lazy drag set on the target object.

Glossary - Q

queue - (1) A linked list of elements waiting to be processed in FIFO order. For example, a queue may be a list of print jobs waiting to be printed. (2) (D of C) A line or list of items waiting to be processed; for example, work to be performed or messages to be displayed.

queued device context - A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context* .

Glossary - R

radio button - (1) A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from a list. Contrast with *check box* . (2) In SAA Advanced Common User Access architecture, a circle with text beside it. Radio buttons are combined to show a user a fixed set of choices from which only one can be selected. The circle is partially filled when a choice is selected.

RAS - Reliability, availability, and serviceability.

raster - (1) In computer graphics, a predetermined pattern of lines that provides uniform coverage of a display space. (T) (2) The coordinate grid that divides the display area of a display device. (A)

read-only file - A file that can be read from but not written to.

real mode - A method of program operation that does not limit or prevent access to any instructions or areas of storage. The operating system loads the entire program into storage and gives the program access to all system resources. Contrast with *protect mode* .

realize - To cause the system to ensure, wherever possible, that the physical color table of a device is set to the closest possible match in the logical color table.

recursive routine - A routine that can call itself, or be called by another routine that was called by the recursive routine.

reentrant - The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase - (1) A word or phrase that is emphasized in a device-dependent manner to inform the user that additional information for the word or phrase is available. (2) (D of C) In hypertext, text that is highlighted and preceded by a single-character input field used to signify the existence of a hypertext link.

reference phrase help - In SAA Common User Access architecture, highlighted words or phrases within help information that a user selects to get additional information.

refresh - To update a window, with changed information, to its current status.

region - A clipping boundary in device space.

register - A part of internal storage having a specified storage capacity and usually intended for a specific purpose. (T)

remote file system - A file-system driver that gains access to a remote system without a block device driver.

resource - The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts,

templates, accelerators, and mnemonics; the definitions are held in a resource file.

resource file - A file containing information used in the definition of a window. Definitions can be of fonts, templates, accelerators, and mnemonics.

restore - To return a window to its original size or position following a sizing or moving action.

retained graphics - Graphic primitives that are remembered by the Presentation Manager interface after they have been drawn. Contrast with *nonretained graphics* .

return code - (1) A value returned to a program to indicate the results of an operation requested by that program. (2) A code used to influence the execution of succeeding instructions.(A)

reverse video - (1) A form of highlighting a character, field, or cursor by reversing the color of the character, field, or cursor with its background; for example, changing a red character on a black background to a black character on a red background. (2) In SAA Basic Common User Access architecture, a screen emphasis feature that interchanges the foreground and background colors of an item.

REXX Language - Restructured Extended Executor. A procedural language that provides batch language functions along with structured programming constructs such as loops; conditional testing and subroutines.

RGB - (1) Color coding in which the brightness of the additive primary colors of light, red, green, and blue, are specified as three distinct values of white light. (2) Pertaining to a color display that accepts signals representing red, green, and blue.

roman - Relating to a type style with upright characters.

root segment - In a hierarchical database, the highest segment in the tree structure.

round-robin scheduling - A process that allows each thread to run for a specified amount of time.

run time - (1) Any instant at which the execution of a particular computer program takes place. (T) (2) The amount of time needed for the execution of a particular computer program. (T) (3) The time during which an instruction in an instruction register is decoded and performed. Synonym for *execution time* .

Glossary - S

SAA - Systems Application Architecture.

SBCS - Single-byte character set.

scheduler - A computer program designed to perform functions such as scheduling, initiation, and termination of jobs.

screen - In SAA Basic Common User Access architecture, the physical surface of a display device upon which information is shown to a user.

screen device context - A logical description of a data destination that is a particular window on the screen. See also *device context* .

SCREEN$ - Character-device name reserved for the display screen.

scroll bar - In SAA Advanced Common User Access architecture, a part of a window, associated with a scrollable area, that a user interacts with to see information that is not currently allows visible.

scrollable entry field - An entry field larger than the visible field.

scrollable selection field - A selection field that contains more choices than are visible.

scrolling - Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window - A window that contains information that is dependent on information in a primary window and is used to supplement the interaction in the primary window.

sector - On disk or diskette storage, an addressable subdivision of a track used to record one block of a program or data.

segment - See *graphics segment* .

segment attributes - Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain - All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonym for *picture chain* .

segment priority - The order in which segments are drawn.

segment store - An area in a normal graphics presentation space where retained graphics segments are stored.

select - To mark or choose an item. Note that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing.

select button - The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor - In SAA Advanced Common User Access architecture, a visual indication that a user has selected a choice. It is represented by outlining the choice with a dotted box. See also *text cursor* .

selection field - (1) In SAA Advanced Common User Access architecture, a set of related choices. See also *entry field* . (2) In SAA Basic Common User Access architecture, an area of a panel that cannot be scrolled and contains a fixed number of choices.

semantics - The relationships between symbols and their meanings.

semaphore - An object used by applications for signalling purposes and for controlling access to serially reusable resources.

separator - In SAA Advanced Common User Access architecture, a line or color boundary that provides a visual distinction between two adjacent areas.

serial dialog box - See *modal dialog box* .

serialization - The consecutive ordering of items.

serialize - To ensure that one or more events occur in a specified sequence.

serially reusable resource (SRR) - A logical resource or object that can be accessed by only one task at a time.

session - (1) A routing mechanism for user interaction via the console; a complete environment that determines how an application runs and how users interact with the application. OS/2 can manage more than one session at a time, and more than one process can run in a session. Each session has its own set of environment variables that determine where OS/2 looks for dynamic-link libraries and other important files. (2) (D of C) In the OS/2 operating system, one instance of a started program or command prompt. Each session is separate from all other sessions that might be running on the computer. The operating system is responsible for coordinating the resources that each session uses, such as computer memory, allocation of processor time, and windows on the screen.

Settings Notebook - A control window that is used to display the settings for an object and to enable the user to change them.

shadow - An object that refers to another object. A shadow is not a copy of another object, but is another representation of the object.

shadow box - The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shared data - Data that is used by two or more programs.

shared memory - In the OS/2 operating system, a segment that can be used by more than one program.

shear - In computer graphics, the forward or backward slant of a graphics symbol or string of such symbols relative to a line perpendicular to the baseline of the symbol.

shell - (1) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices, and touch-sensitive screens, and communicate them to the operating system. (2) Software that allows a kernel program to run under different operating-system environments.

shutdown - The process of ending operation of a system or a subsystem, following a defined procedure.

sibling processes - Child processes that have the same parent process.

sibling windows - Child windows that have the same parent window.

simple list - A list of like values; for example, a list of user names. Contrast with *mixed list* .

single-byte character set (SBCS) - A character set in which each character is represented by a one-byte code. Contrast with *double-byte character set* .

slider box - In SAA Advanced Common User Access architecture: a part of the scroll bar that shows the position and size of the visible information in a window relative to the total amount of information available. Also known as *thumb mark* .

SOM - System Object Model.

source file - A file that contains source statements for items such as high-level language programs and data description specifications.

source statement - A statement written in a programming language.

specific dynamic-link module - A dynamic-link module created for the exclusive use of an application.

spin button - In SAA Advanced Common User Access architecture, a type of entry field that shows a scrollable ring of choices from which a user can select a choice. After the last choice is displayed, the first choice is displayed again. A user can also type a choice from the scrollable ring into the entry field without interacting with the spin button.

spline - A sequence of one or more Bézier curves.

spooler - A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete and the required device is available. The spooler prevents output from different sources from being intermixed.

stack - A list constructed and maintained so that the next data element to be retrieved is the most recently stored. This method is characterized as last-in-first-out (LIFO).

standard window - A collection of window elements that form a panel. The standard window can include one or more of the following window elements: sizing borders, system menu icon, title bar, maximize/minimize/restore icons, action bar and pull-downs, scroll bars, and client area.

static control - The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

static storage - (1) A read/write storage unit in which data is retained in the absence of control signals. (A) Static storage may use dynamic addressing or sensing circuits. (2) Storage other than *dynamic storage* . (A)

style - See *window style* .

subclass - A class that inherits from another class. See also *Inheritance* .

subdirectory - In an IBM personal computer, a file referred to in a root directory that contains the names of other files stored on the diskette or fixed disk.

superclass - A class from which another class inherits. See also *inheritance* .

swapping - (1) A process that interchanges the contents of an area of real storage with the contents of an area in auxiliary storage. (I) (A) (2) In a system with virtual storage, a paging technique that writes the active pages of a job to auxiliary storage and reads pages of another job from auxiliary storage into real storage. (3) The process of temporarily removing an active job from main storage, saving it on disk, and processing another job in the area of main storage formerly occupied by the first job.

switch - (1) In SAA usage, to move the cursor from one point of interest to another; for example, to move from one screen or window to another or from a place within a displayed image to another place on the same displayed image. (2) In a computer program, a conditional instruction and an indicator to be interrogated by that instruction. (3) A device or programming technique for making a selection, for example, a toggle, a conditional jump.

switch list - See *Task List* .

symbolic identifier - A text string that equates to an integer value in an include file, which is used to identify a programming object.

symbols - In Information Presentation Facility, a document element used to produce characters that cannot be entered from the keyboard.

synchronous - Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals. (T) See also *asynchronous* .

System Menu - In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

System Object Model (SOM) - A mechanism for language-neutral, object-oriented programming in the OS/2 environment.

system queue - The master queue for all pointer device or keyboard events.

system-defined messages - Messages that control the operations of applications and provides input an other information for applications to process.

Systems Application Architecture (SAA) - A set of IBM software interfaces, conventions, and protocols that provide a framework for designing and developing applications that are consistent across systems.

Glossary - T

table tags - In Information Presentation Facility, a document element that formats text in an arrangement of rows and columns.

tag - (1) One or more characters attached to a set of data that contain information about the set, including its identification. (I) (A) (2) In Generalized Markup Language markup, a name for a type of document or document element that is entered in the source document to identify it.

target object - An object to which the user is transferring information.

Task List - In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

terminate-and-stay-resident (TSR) - Pertaining to an application that modifies an operating system interrupt vector to point to its own location (known as hooking an interrupt).

text - Characters or symbols.

text cursor - A symbol displayed in an entry field that indicates where typed input will appear.

text window - Also known as the VIO window.

text-windowed application - The environment in which the operating system performs advanced-video input and output operations.

thread - A unit of execution within a process. It uses the resources of the process.

thumb mark - The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a *slider box* .

thunk - Term used to describe the process of address conversion, stack and structure realignment, etc., necessary when passing control between 16-bit and 32-bit modules.

tilde - A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

time slice - (1) An interval of time on the processing unit allocated for use in performing a task. After the interval has expired, processing-unit time is allocated to another task, so a task cannot monopolize processing-unit time beyond a fixed limit. (2) In systems with time sharing, a segment of time allocated to a terminal job.

time-critical process - A process that must be performed within a specified time after an event has occurred.

timer - A facility provided under the Presentation Manager, whereby Presentation Manager will dispatch a message of class WM_TIMER to a particular window at specified intervals. This capability may be used by an application to perform a specific processing task at predetermined intervals, without the necessity for the application to explicitly keep track of the passage of time.

timer tick - See *clock tick* .

title bar - In SAA Advanced Common User Access architecture, the area at the top of each window that contains the window title and system menu icon. When appropriate, it also contains the minimize, maximize, and restore icons. Contrast with *panel title* .

TLB - Translation lookaside buffer.

transaction - An exchange between a workstation and another device that accomplishes a particular action or result.

transform - (1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a *transformation* .

Translation lookaside buffer (TLB) - A hardware-based address caching mechanism for paging information.

Tree - In the Presentation Manager, the window in the *File Manager* that shows the organization of drives and directories.

truncate - (1) To terminate a computational process in accordance with some rule (A) (2) To remove the beginning or ending elements of a string. (3) To drop data that cannot be printed or displayed in the line width specified or available. (4) To shorten a field or statement to a specified length.

TSR - Terminate-and-stay-resident.

Glossary - U

unnamed pipe - A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with *named pipe* .

unordered list - In Information Presentation Facility, a vertical arrangement of items in a list, with each item in the list preceded by a special character or bullet.

update region - A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window that are visually invalid or incorrect, and therefore are in need of repainting.

user interface - Hardware, software, or both that allows a user to interact with and perform operations on a system, program, or device.

User Shell - A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

utility program - (1) A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program. (T) (2) A program designed to perform an everyday task such as copying data from one storage device to another. (A)

Glossary - V

value set control - A visual component that enables a user to select one choice from a group of mutually exclusive choices.

vector font - A set of symbols, each of which is created as a series of lines and curves. Synonymous with *outline font* . Contrast with *image font* .

VGA - Video graphics array.

view - A way of looking at an object's information.

viewing pipeline - The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window - A clipping boundary that defines the visible part of model space.

VIO - Video Input/Output.

virtual memory (VM) - Synonymous with *virtual storage* .

virtual storage - (1) The storage space that may be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations. (I) (A) (2) Addressable space that is apparent to the user as the processor storage space, from which the instructions and the data are mapped into the processor storage locations. (3) Synonymous with *virtual memory* .

visible region - A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

volume - (1) A file-system driver that uses a block device driver for input and output operations to a local or remote device. (I) (2) A portion of data, together with its data carrier, that can be handled conveniently as a unit.

Glossary - W

wildcard character - Synonymous with *global file-name character* .

window - (1) A portion of a display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. (A) (2) An area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen.

window class - The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates - A set of coordinates by which a window position or size is defined; measured in device units, or *pels* .

window handle - Unique identifier of a window, generated by Presentation Manager when the window is created, and used by applications to direct messages to the window.

window procedure - Code that is activated in response to a message. The procedure controls the appearance and behavior of its associated windows.

window rectangle - The means by which the size and position of a window is described in relation to the desktop window.

window resource - A read-only data segment stored in the .EXE file of an application o the .DLL file of a dynamic link library.

window style - The set of properties that influence how events related to a particular window will be processed.

window title - In SAA Advanced Common User Access architecture, the area in the title bar that contains the name of the application and the OS/2 operating system file name, if applicable.

Workplace Shell - The OS/2 object-oriented, graphical user interface.

workstation - (1) A display screen together with attachments such as a keyboard, a local copy device, or a tablet. (2) (D of C) One or more programmable or nonprogrammable devices that allow a user to do work.

world coordinates - A device-independent Cartesian coordinate system used by the application program for specifying graphical input and output. (I) (A)

world-coordinate space - Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG - What-You-See-Is-What-You-Get. A capability of a text editor to continually display pages exactly as they will be printed.

Glossary - X

There are no glossary terms for this starting letter.

Glossary - Y

There are no glossary terms for this starting letter.

Glossary - Z

z-order - The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming - The progressive scaling of an entire display image in order to give the visual impression of movement of all or part of a display group toward or away from an observer. (I) (A)

8.3 file-name format - A file-naming convention in which file names are limited to eight characters before and three characters after a single dot. Usually pronounced "eight-dot-three." See also *non-8.3 file-name format* .
